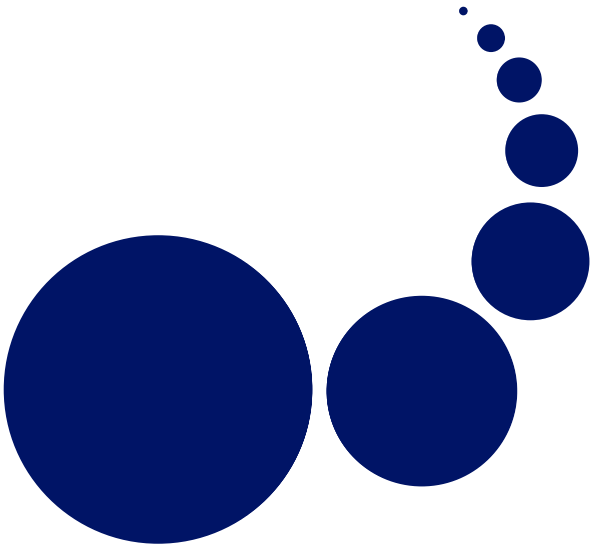
SCALABLE COMPUTING
Practice and Experience



Volume 6, Number 2, July 2005

ISSN 1895-1767

SWPS

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 6, Number 2, July 2005

TABLE OF CONTENTS

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. iii–iv. http://www.scpe.org
ISSN 1895-1767
© 2005 SWPS

EDITORIAL: BUILDING GRID COMMUNITIES

There has been a significant increase in interest in Grid Computing in the last year—from the computer science community, the platform vendors, and interestingly, from the application scientists (Physicists, Biologists, etc). Significantly, Grid computing emphasises the same challenges in interoperability, security, fault tolerance, performance and data management as the distributed computing community—albeit grounded with specific application scenarios from the science and engineering communities. A key aspect of Grid computing is the need to couple computational and data resources to form "Virtual Organisations" (VOs), via resource scheduling and discovery approaches. A VO in this context is created by combining capability across a number of different administrative domains, to run a single large problem. Hence a single application program may execute tasks on multiple resources—using the concept of a "SuperScheduler". The SuperScheduler does not own any resources itself, but connects to a number of different local resource scheduling systems, within different geographically distributed administrative domains. Choosing suitable resources on which to schedule operations has generally involved a discovery process utilising a search of registry services to locate resources of interest. The mechanisms used within this process have been quite limited in scope, offering a limited set of queries to interrogate a (generally) LDAP based repository (via `grid-info-search` in Globus for instance).

Having an efficient discovery mechanism within a Grid environment is quite significant—as it provides the infrastructure needed to support dynamic and fault tolerant VOs. The discovery mechanism also needs to be efficient—as searching for suitable resources—or more recently "services" (whereby access to all computational and data resources is seen as a computational or data service) – requires querying across distributed registries. The discovery process can also be seen as a way to establish dynamic relationships between Grid services—persistent or transient—as the discover process can be seen as a first stage in establishing an association with another service. Hence, when initiating a discovery process, a service user should identify the type of association to be formed with a service provider. Examples of such associations may be client/server (generally), or Peer-2-Peer. Viewed in this way, service discovery may be undertaken "passively"—similar to a service lookup in a registry, or "actively" when the discovery mechanism is intended to form a particular type of association with another service—and based on parameters such as cost, performance, Quality of Service (QoS), or trust.

Groups. As the number of services on the Grid increase, so will the potential interactions between these services. This can lead to high traffic volumes on the network, and potentially act as a barrier to scalability. The use of the "group" paradigm is significant in this context, to limit interaction within a small community of services (in the first instance). Service groups can be formed based on a number of different criteria, ranging from geographical location of services, by service types—such as mathematical services, graphics services, data analysis services etc, service ownership, service costs, or service priorities. Members of a group can interact with each other more efficiently (via multicast messages, for instance), and can load balance requests sent to the group. The concept of forming a group or community also implies some level of trust between the members, implying the availability of efficient mechanisms to share state with other members of the group. The provision of shared memory based abstractions become necessary in this context, as group members may need to repeatedly share state, and be alerted as new members enter/leave the group. An important consideration in this context is the ability to decide which services should be allowed within a group—and how the group structure should evolve. Different service behaviours and roles can co-exist within a group—even though the services are targeted at a particular application domain. Therefore, in a group of mathematical services, one could have broker services, service users, community management services, and monitoring services. Some of these are required to manage and maintain the group itself, whilst others provide specialised capability.

The group idea also leads to the formation of "small world" networks—primarily interaction networks with a small average path length between members, and a large internal connectivity (i. e. a clustering coefficient that is independent of network size). These networks are likely to be of great significance in scientific computing—as they closely model data sharing between scientists within a given domain (such as genomics, astronomy, etc)—and different from sharing music data in systems like Gnutella. Once a small world network has been established, various assumptions about other members in the group may be made—such as their ability to provide particular types of data (or their speciality), trust that can be placed in them, and their ability to respond in a timely fashion. Identifying scenarios where such networks may be established, and subsequently providing suitable middleware to sustain these, will be important in Grid systems. One member within such a small world network (group) may be allocated the role of being a "cluster head"—thereby acting as a gateway to other small world networks. Hence a federation of such small world networks may be established, that span domain and organisational boundaries.

The Importance of Shared Semantics. Establishing a group/community of services also necessitates the description of common semantics. Such a description should allow roles of each member to be defined, along with specialist services being offered by them, and provide a mechanism for communication between the members. Hence, two types of descriptions are necessary: (1) those that are independent of any particular application domain, and provide a means to define roles, and (2) those that are specific to particular application domains, and services that need to be provided within that domain. Significant progress has been made towards describing attributes of computational and data resources in a unified way—so that a query to a SuperScheduler could be understood by multiple resource managers. Little progress however has been made towards standardising descriptions of services within particular application domains—a time consuming and consensus building process within a scientific community. Both of these descriptions are necessary to enable multi-disciplinary science and engineering—and to enable a better understanding of common requirements of all of these communities. An important concern is the ability of these different data models to interact—as there is unlikely to be a single model used by all within or across communities. Resolving differences in representation, or providing support for automated negotiation between members to resolve semantic differences, become important services that need to be supported within a group.

Although significant progress has been made in Grid computing, the next phase will require more efficient mechanisms to organise and coordinate participants (individuals or institutions) within groups. Mechanisms to support the formation of groups are also significant—i. e. identifying which members should belong to which group, and how their membership can be sustained. Current emphasis has been on infrastructure installation (such as high speed networks) and means to provide common interfaces to resource management systems. To enable more effective interaction between services which run on this infrastructure, standardisation of common data models becomes significant. The Grid should eventually provide the computational and data infrastructure necessary to enable groups of scientists to undertake multi-disciplinary work. It should also provide the necessary entry points for connecting resources with a range of different capabilities—such as high end computational clusters and parallel machines, to sensor networks capable of data capture at source. An important aspect of Grid computing has been the significant interactions between the computer science and the application science communities—and for Grid computing to mature, these need to be strengthened further.

Acknowledgement. Thanks to José C. Cunha of the CITI Centre, New University of Lisbon (Portugal), for interesting discussions on Groups.

Omer F. Rana,
Cardiff University
and the Welsh E-Science/Grid Computing Centre, UK

Scalable Computing: Practice and Experience
Volume 6, Number 2, p. v. `http://www.scpe.org`
ISSN 1895-1767
© 2005 SWPS

GUEST EDITORS' INTRODUCTION

A selection of best papers of the 4th Austrian-Hungarian Workshop on Distributed and Parallel Systems are presented in this issue. The series of workshops started as a local meeting in 1992 and it grew to an internationally acclaimed event of computing techniques covering not just parallel and distributed programming in the classical sense but also emerging topics like ubiquitous and pervasive computing, cluster and grid technology, multimedia and challenging applications.

Thoai et al focus on a fundamental problem of parallel program development: debugging. Since the execution of parallel programs is nondeterministic, forcing a certain execution trace among the many possible ones requires sophisticated replay techniques. The paper presents how the probe effect can be eliminated and the overhead can be minimised during replay based debugging.

The paper by Lovas et al. presents parallel implementation of an ultra-short range weather prediction method, supported by a graphical development tool, P-GRADE. The paper introduces all stages of the program development from editing to performance analysis.

A novel approach for resource discovery is presented by Juhász et al. A grid system should be able to provide brokering services for potentially thousands and millions of resources. Most approaches nowadays are either centralised or flat and in such a way are not really scaleable. The paper proposes a hierarchical fault-tolerant multicast discovery scheme.

The paper by Heinzlreiter et al. presents a grid middleware that enables realising novel interactive visualisation systems in grid environments. Tools like the Grid Visualisation Kernel (GVK) are leading towards pioneering grid based Virtual Reality applications.

Fault diagnosis is the central issue of the paper by Polgár et al. They propose a modification of P-graph model in order to improve fault diagnosis in complex multiprocessor systems.

Bósa et al. introduce advanced fault tolerating mechanisms for Distributed Maple, a parallel computer algebra system. The tolerance of failed links and nodes is enhanced by adding reconnection and restart mechanisms as well as change the virtual root node in order to avoid overall failure.

Emerging mobile applications raise the issue of context awareness. Ferscha et al. introduce the techniques related to context sensing, representation and delivery and proposes a new approach for context based mobile computing.

Goldschmidt et al. analyse the requirements of adaptive multimedia servers where the dynamic migration of multimedia applications is supported and an agent-based infrastructure is proposed. The idea is supported by a specification and implementation of a CORBA-based interface.

The progress of multimedia over the internet is obvious that raises the need of intelligent video caches. The paper by Schojer et al. introduces a proxy cache that allows fast and efficient adaptation of video based on the MPEG-4 standard.

These papers demonstrate the wide spectrum of the workshop topics: from distributed computing via grids towards novel complex systems.

Zsolt Németh
Dieter Kranzlmüller
Péter Kacsuk
Jens Volkert

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 1–12. http://www.scpe.org
ISSN 1895-1767
© 2005 SWPS

AN APPROACH TO PROVIDING SMALL-WAITING TIME DURING DEBUGGING MESSAGE-PASSING PROGRAMS

NAM THOAI[*] AND JENS VOLKERT[†]

Abstract. Cyclic debugging, where a program is executed repeatedly, is a popular methodology for tracking down and eliminating bugs. Breakpointing is used in cyclic debugging to stop the execution at arbitrary points and inspect the program's state. These techniques are well understood for sequential programs but they require additional efforts when applied to parallel programs. For example, record&replay mechanisms are required due to nondeterminism. A problem is the cost associated with restarting the program's execution every time from the beginning until arriving at the breakpoints. A corresponding solution is offered by combining checkpointing and debugging, which allows restarting an execution at an intermediate state. However, minimizing the replay time is still a challenge. Previous methods either cannot ensure that the replay time has an upper bound or accept the probe effect, where the program's behavior changes due to the overhead of additional code. Small waiting time is the key that allows to develop debugging tools, in which some degree of interactivity for the user's investigations is required. This paper introduces the MRT method to limit the waiting time with low logging overhead and the four-phase-replay method to avoid the probe effect. The resulting techniques are able to reduce the waiting time and the costs of cyclic debugging.

Key words. Parallel debugging, checkpointing, message logging, replay time, probe effect

1. Introduction.

Debugging is an important part of software engineering. Obviously, a program is only valid if it runs correctly. A popular traditional method in this area is cyclic debugging, where a program is run repeatedly to collect more information about its intermediate states and finally to locate the origin of errors. A related technique often used in cyclic debugging is breakpointing. It allows programmers to stop and examine a program at interesting points during execution.

Parallel architectures and parallel programming have become the key to solve large-scale problems in science and engineering today. Thus, developing a debugging mechanism for parallel programs is important and necessary. Such a solution should support cyclic debugging with breakpointing in parallel programs. However, in parallel programs, communication and synchronization may introduce nondeterministic behavior. In this case, consecutive runs with the same input data may yield different executions and different results. This effect causes serious problems during debugging, because subsequent executions of a program may not reproduce the original bugs, and cyclic debugging is ad hoc not possible.

To overcome this irreproducibility effect, several record&replay techniques [11, 15, 23] have been proposed. These techniques are based on a two-step approach. The first step is a *record phase*, in which data related to nondeterministic events are stored in trace files. Afterwards, these trace data are used as a constraint for the program during subsequent *replay phases* to produce equivalent executions.

A major problem of this approach is the waiting time because programs are always re-started from the beginning. Especially with long-running parallel programs, where execution times of days, weeks, or even months are possible, a re-starting point at the beginning is unacceptable. Starting programs from intermediate states can solve this problem. Such a solution is offered by *Incremental Replay* techniques [31]. They support to start a parallel program at intermediate points and investigate only a part of one process at a time. As an extension, the goal of our approach is to stop at an arbitrary distributed breakpoint and to initiate re-execution of the program in minimum time and with a low overhead.

Requirement to construct adequate recovery lines on multiple processes was previously described in literature on fault tolerance computing as well as debugging [1, 5, 8]. The restriction of these methods is that recovery lines, which are global states that the program can be restarted, are only established at consistent global checkpoints, because inconsistent global states prohibit failure-free execution. Therefore, limiting the rollback distance, which is the distance between the recovery line and the corresponding distributed breakpoint, is impossible. Some additional techniques allow to shorten the rollback distance but the associated overhead may be rather high because many checkpoints are required and a lot of messages must be logged [28]. Both represent serious obstacles during developing debugging tools, which must provide some degree of interactivity for the user's investigations.

[*]Faculty of Information Technology, HCMC University of Technology, 268 Ly Thuong Kiet Street, District 10, Ho Chi Minh City, Vietnam (nam@dit.hcmut.edu.vn).

[†]GUP Linz, Johannes Kepler University Linz, Altenbergerstr. 69, A-4040 Linz, Austria/Europe (volkert@gup.uni-linz.ac.at).

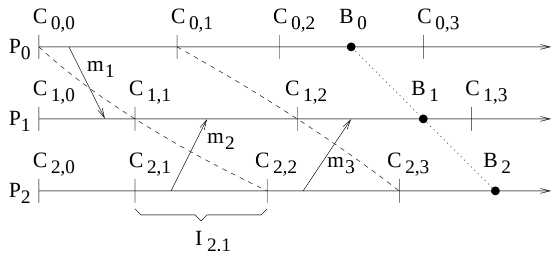


FIG. 2.1. *Event graph and checkpoint events.*

To solve the above problem, a new replay technique has been developed: the *Shortcut Replay* method [29] allows to construct flexible recovery lines and thus the rollback distance is shortened. However, the overhead of message logging is still high. Therefore, the trade-off between the rollback distance and the message-logging overhead should be examined. The Rollback-One-Step method (ROS) [28] is an effort to address this problem by establishing an upper bound of the rollback distance. However, the replay time depends on the number of processes and may be rather long with large-scale, long-running parallel programs (as discussed in Section 5.2). A new method, named MRT (the Minimizing the Replay Time method), is presented in this paper. It ensures the upper bound for the replay time, which is independent of the number of processes. An implementation of MRT and its result demonstrate the efficiency of this approach.

Another important point in debugging is that the observation should not affect the actual behavior of the program. However, the instrumented code and checkpointing activities may substantially affect the behavior of the program. Therefore, the four-phase-replay method is offered as a replay method for MRT with low overhead.

This paper is divided into 10 sections. Basic definitions of parallel programs, the event graph and checkpointing are described in Section 2. Definition of distributed breakpoint is introduced in Section 3. Section 4 explains the reason of nondeterministic behavior of parallel programs and provides an overview of record&replay methods to solve the nondeterminism problem. The definition of the rollback/replay distance is also given in Section 4. ROS and its limitations are shown in Section 5. After that, MRT is introduced in Section 6 and its implementation is described in Section 7. The four-phase-replay method to avoid the probe effect is presented in Section 8. A comparison between MRT using the four-phase-replay method and other methods as well as the future work to integrate MRT into the Process Isolation technique are discussed in Section 9. The paper finishes with conclusions in Section 10.

2. System model. The parallel programs considered in this paper are message-passing programs, which include n processes $P_0, P_1, \ldots, P_{n-1}$ that exchange data through messages. To model a program's execution, the event graph model [11] is utilized. This model describes the interesting operations in all processes and their relations. An event graph is a directed graph $G = (E, \rightarrow)$, where the non-empty set of events E is comprised of the events $e_{p,i}$ of G observed during program execution, with i denoting the sequential order on process P_p. The relations between events are "happened-before" relations [14]. Let $e_{p,i} \rightarrow e_{q,j}$ denote that $e_{p,i}$ is happened-before $e_{q,j}$ and there is an edge from event $e_{p,i}$ to event $e_{q,j}$ in G with the "tail" at event $e_{p,i}$ and the "head" at event $e_{q,j}$.

In our message-passing programs, the event set E contains two kinds of event. The first kind are communication events. Only events, which concern sending and delivering of messages, are interesting. They are called send and receive events, respectively. In order to obtain these events for a particular program run, the program's source code is instrumented and re-execution is initiated. The events (and corresponding data) are stored in trace files.

The second kind of events are checkpointing events, which represent local checkpoints. A local checkpoint is the local state of a process at a particular point in time. The i-th local checkpoint taken by process P_p is denoted by $C_{p,i}$. We assume that process P_p takes an initial checkpoint $C_{p,0}$ immediately before execution begins, and ends with a virtual checkpoint that represents the last state attained before termination. The i-th checkpoint interval of process P_p, denoted by $I_{p,i}$, includes all the events that happened on process P_p between checkpoint event $C_{p,i}$ and checkpoint event $C_{p,i+1}$, including $C_{p,i}$ but not $C_{p,i+1}$. The maximum execution time of all checkpoint intervals during the initial execution is denoted by T.

A global checkpoint is a set of local checkpoints, one from each process. When considering a global checkpoint GC, two categories of messages are particularly important: messages that have been delivered in GC, although the corresponding send events occur only after the local checkpoints comprising GC (orphan messages) and messages that have been sent but not delivered in GC (in-transit messages). A global checkpoint is consistent if there are no orphan messages with respect to it [8]. An inconsistent global checkpoint is a global checkpoint which is not consistent.

For example, in Fig. 2.1, $C_{0,0}$, $C_{0,1}$, $C_{1,2}$, $C_{2,0}$, etc. are local checkpoints, while $(C_{0,0}, C_{1,1}, C_{2,2})$ and $(C_{0,1}, C_{1,2}, C_{2,3})$ are global checkpoints. Messages m_2 and m_3 are in-transit messages of $(C_{0,0}, C_{1,1}, C_{2,2})$ and $(C_{0,1}, C_{1,2}, C_{2,3})$, respectively. Message m_1 is an orphan message of $(C_{0,0}, C_{1,1}, C_{2,2})$. Therefore $(C_{0,0}, C_{1,1}, C_{2,2})$ is inconsistent, while $(C_{0,1}, C_{1,2}, C_{2,3})$ is consistent.

3. Breakpointing with the event graph. Breakpointing allows programmer to halt and examine a program at interesting points during execution. More precisely, it gives a debugger the ability to suspend the debuggee when its thread of control reaches a particular point. The program's stack and data values can then be examined, data values possibly modified, and program execution continued until the program encounters another breakpoint location, fault, or it terminates.

Setting breakpoints in sequential programs is well understood. There is only one thread of computation and thus the execution of the thread should be stopped when the breakpoint is hit. It is more complex in parallel programs since several threads or processes exist concurrently and thus different effects may happen when hitting a breakpoint. According to how many processes will be stopped by hitting a breakpoint, Kacsuk classifies several kinds of breakpoints such as local breakpoint, message breakpoint, general global breakpoint set, and collective breakpoint set [9]. In this paper, distributed breakpoints are used. A distributed breakpoint is a set of breakpoints, each breakpoint on each process, and a breakpoint only stops its local process. Note that there is no happened-before relation [14] between any pair of breakpoints belonging to a distributed breakpoint. This kind of breakpoints can be compared to (strongly complete) global breakpoint set in the classification of Kacsuk [9]. For example, in Fig. 2.1, (B_0, B_1, B_2) is a distributed breakpoint; and processes P_0, P_1, and P_2 will stop at B_0, B_1, and B_2 respectively.

There are two ways to establish a distributed breakpoint: (1) users will manually locate local breakpoints on all processes, and (2) the distributed breakpoint is generated automatically. The advantage of the first method is that users can stop the execution at any interesting point during its execution. However, it requires that the chosen distributed breakpoint must be consistent but this condition is difficult to achieve if users do not know the relations between events on the running program. The event graph model can be used in this case, but it also takes users a lot of time to track the relations.

In addition, users may want to examine the interference of other processes on one process or a group of processes at some points. The same idea is given in the causal distributed breakpoints [7], which restores each process to the earliest state that reflects all events that are happened-before the breakpoint. It is constructed based on a breakpoint in the breakpoint process as follows:

1. The breakpoint process is stopped at the well-defined breakpoint, and
2. Other processes are stopped at the earliest states that reflect all events in that processes that are happened-before the breakpoint event.

The conventional notion of a breakpoint in a sequential program can be kept in parallel programs through causal distributed breakpoints. Based on the event graph, the causal distributed breakpoints are constructed easily since relations between events can be obtained through the event graph.

4. Nondeterminism of parallel programs and solutions.

4.1. Nondeterminism and the irreproducibility effect. One hindrance for cyclic debugging of parallel programs is nondeterministic behavior, where a program may run with different paths and produce different results in subsequent executions with the same input. There are many reasons that lead to nondeterministic behavior of a program. Random number generators are a simple example. Obviously, a random number generator gives different values in different executions. Other sources are return values from system calls such as *gettimeofday*(), *getpid*(), etc. These can be seen in both sequential and parallel programs. In message-passing programs, an additional source is the order of incoming messages at wild-card receives, which are supported in most communication libraries. If a wild-card is used in a receive operation, the process accepts a message from any source. An example wild-card is MPI_ANY_SOURCE in MPI programs [19]. The different orders of incoming messages may come from processor speed, load imbalance, scheduling decisions of the processor and the operating system, network throughput and network conflicts, etc.

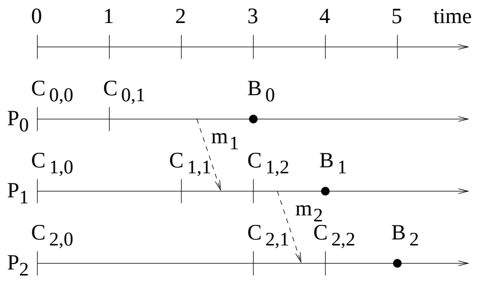


FIG. 4.1. *The rollback/replay distance.*

Due to nondeterminism, subsequent executions of the parallel program with the same input may not reproduce the original bugs. This effect is called irreproducibility effect [25] or non-repeatability effect [20]. It may cause programmers confusion due to the disappearance of certain bugs and the appearance of other new bugs in repeated debugging cycles.

4.2. Record&replay methods. Record&replay methods are proposed to solve the problem of the irreproducibility effect. There are two steps in these approaches. The first step detects nondeterministic events and stores data related to them to trace files. This step is called the "record phase". During the second step, the trace data are used as a constraint for the program, called the "replay phase", to produce equivalent executions.

Record&replay methods can be classified into two categories: data-driven [11] (or content-based/data-based [23]) and control-driven [11] (or ordering-based [23]). In data-driven, the contents of each message are recorded when they are received by the corresponding processes. Such a method is proposed in [4]. The biggest drawback of this technique is the requirement of significant monitor and storage overhead. Furthermore, it does not show the interactions between the different processes, and thus hinders the task of finding the cause of a bug [15]. However, it is still useful for tracing I/O or for tracing the result of certain system calls such as $gettimeofday()$ or $random()$.

The control-driven methods are based on the piecewise deterministic (PWD) execution model [26]: process execution is divided into a sequence of state intervals, each of which is started by a nondeterministic event. The execution within an interval is completely deterministic. Under the PWD assumption, execution of each deterministic interval depends only on the sequence of nondeterministic events that preceded the interval's beginning. Thus the equivalent executions are ensured if the ordering of nondeterministic operations on all processes is the same as in the initial execution. Such an approach of control-driven technique is *Instant Replay* [15], which can be applied for both shared memory and message-passing programs. Following the PWD execution model, if each process is given the same input values in the same ordering during the successive executions, it will produce the same behavior each time[1]. Consequently, each process will produce the same output values in the same order. These output values may then serve as input values for other processes. Therefore, we need only to trace the relative order of significant events instead of the data associated with these events. The advantage of this technique is that it requires less time and space to save the information needed for replay. Several solutions based on control-driven are implemented for both PVM programs [17] and MPI programs [2, 10].

To improve the control-driven replay technique, an optimal tracing and replay method is proposed in [20]. The key technique is that only events affecting the race conditions have to be traced. A race condition in message-passing programs occurs, if two or more messages are simultaneously to arrive at a particular receive operation and each of them can be accepted first.

4.3. Replay time and rollback/replay distance. The rollback/replay distance is introduced in [28]. The running time from event e_1 to event e_2 on the same process is called the distance between e_1 and e_2, denoted $d(e_1, e_2)$. The distance between global states $G_1 = (g_{1,0}, g_{1,1}, \ldots, g_{1,n-1})$ and $G_2 = (g_{2,0}, g_{2,1}, \ldots, g_{2,n-1})$ is:

$D = \max(d_i)$ where $d_i = d(g_{1,i}, g_{2,i})$ and $0 \le i \le n-1$

Note that the definition of the distance between global states G_1 and G_2 is only valid if $g_{1,i} \rightarrow g_{2,i}$ or $g_{1,i} = g_{2,i}$ (for all i: $0 \le i \le n-1$), where "$\rightarrow$" is Lamport's "*happened before*" relation [14]. For example, in Fig. 4.1,

[1]Input values are the contents of messages received or the values of shared memory locations referenced.

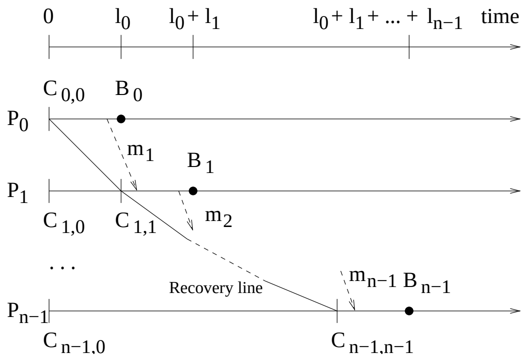


Fig. 4.2. *The upper bound of the replay time.*

the distance between $C_{0,1}$ and B_0 is 2. The distance between $(C_{0,0}, C_{1,1}, C_{2,1})$ and (B_0, B_1, B_2) is 3 and the distance between $(C_{0,1}, C_{1,2}, C_{2,2})$ and (B_0, B_1, B_2) is 2. A relation between the rollback/replay distance and the replay time is described in Theorem 1.

Theorem 1. *If the rollback/replay distance has an upper bound L, an upper bound of the replay time is n.L, where n is number of processes.*

Proof. The worst case is that each process waits for messages from another process and creates a waiting chain of processes as in Fig. 4.2. In this case, jobs are mostly executed sequentially. Thus the maximum replay time is $\sum_{i=0}^{n-1} L_i$ (L_i is the replay distance on process P_i). This value is less than $n.L$. □

In this paper, we are interested in the distance between the recovery line and the corresponding distributed breakpoint, which is called the rollback distance or the replay distance. This replay distance is used to estimate the replay time. They are different because the replay distance is based on the previous execution while the replay time is determined during re-execution. Fig. 4.1 is an example of this difference. If users want to stop at (B_0, B_1, B_2) while a program is recovered at $(C_{0,0}, C_{1,1}, C_{2,1})$, then the rollback/replay distance is 3 but the replay time is approximately 4 due to waiting of messages m_1 and m_2. The replay time is often larger than the replay distance due to the required waiting of messages.

All record&replay methods described in Section 4.2 allow a parallel program to be re-executed deterministically from the beginning state. Of course, the execution time from the beginning state to the distributed breakpoint is obviously long if the distributed breakpoint is set far from the beginning state in large-scale, long-running parallel programs. To solve this problem, checkpointing can be used. An example is *Incremental Replay* [31]. This replay technique uses checkpointing to allow users to run any part (checkpoint interval) of a process immediately. By using checkpointing, users do not wait for the process to run from the beginning state, thus reducing the waiting time. In addition, it also provides bounded-time in replay processes [31], which means that the replay time of a checkpoint interval does not exceed a permitted limit. Cyclic debugging is more useful when using Incremental Replay. However, Incremental Replay only supports users to replay one checkpoint interval of one process each time. (The other processes are neglected.) Each checkpoint can be seen as a breakpoint. Programmers can reach any breakpoint on a process immediately but they cannot examine the interactions between processes. In other words, Incremental Replay prohibits the use of distributed breakpoints.

To minimize the waiting time during debugging, the program can be restarted at an intermediate state by using checkpointing and rollback-recovery methods. Many checkpointing and rollback-recovery methods are proposed in fault tolerance and debugging areas [1, 5, 8, 26]. However, the rollback/replay distance is still rather long in some cases [29]. It prohibits developing debugging tools, which must provide some degree of interactivity for the user's investigations.

5. ROS-Rollback-One-Step checkpointing.

5.1. Characteristics of ROS.

ROS [28] is the first effort to minimize the replay time. In this method, recovery lines can be established at either consistent or inconsistent global checkpoints. It differs from other methods, in which only consistent global checkpoints are chosen as recovery lines. In order to produce correct executions even if an inconsistent global checkpoint is used, *Shortcut Replay* [29] is used. The key technique used in Shortcut Replay is *bypassing orphan messages.* This means that orphan messages are detected and ignored in re-execution based on trace data. Therefore, the re-execution can enforce the same event ordering as

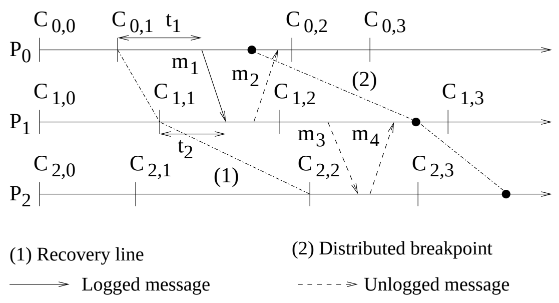


FIG. 5.1. *The replay time in Rollback-One-Step (ROS).*

observed during the record phase. This technique opens a possibility for minimizing the replay time in contrast to former replay techniques.

The bypassing orphan messages technique allows to minimize the rollback/replay distance between the recovery line and the corresponding distributed breakpoint. ROS ensures that an upper bound of the rollback/replay distance is $2T$ [28]. Another advantage of this method is that only a small number of messages needs to be logged. Results of an implementation of this method show that the number of logged messages is mostly less than 5%, which underlines the efficiency of the logging algorithm [28].

5.2. Replay time in ROS. An upper bound of the replay distance in ROS is $2T$ so that an upper bound of the replay time is $2nT$, where n is number of processes, following Theorem 1. The upper bound can be lowered to nT, if an additional logging rule is applied. This requires that the incoming message must be logged if the time elapsed since the last checkpoint on the send process is larger than the one on the receiving process. For example, in Fig. 5.1, message m_1 must be logged due to $t_1 > t_2$. This implies that a process does not wait for messages sent from another process if they are started at checkpoints with the same index. In addition, messages from $I_{p,i}$ to $I_{q,j}$ with $i < j$ are logged[2] so that the worst case of ROS is shown in Fig. 5.1. All processes are rolled back one step except process P_0[3]; and if the recovery checkpoint in process P_k $(k > 0)$ is $C_{k,i}$, then (1) the recovery checkpoint in process P_{k+1} is $C_{k+1,i+1}$ and (2) there is a message from $I_{k,i+1}$ to $I_{k+1,i+1}$, which is not logged (message m_3 in Fig. 5.1). The replay time is only nT, where n is number of processes. In either case, the upper bound of the replay time depends on the number of processes and thus it may be long if the number of processes is large.

6. MRT-Minimizing the Replay Time. MRT is an extension of ROS. This new method tries to keep advantages of the former method and ensures that the upper bound of the replay time is independent of the number of processes. The checkpointing techniques used in both methods are the same. This means that the state of each process is stored periodically in stable storage and the current checkpoint interval index is piggybacked on the transferred messages. When process P_p receives a message m in interval $I_{p,i}$ with the checkpoint interval index j piggybacked on m such that $j > i$, a new checkpoint with index j is immediately taken. Furthermore, the checkpoint must be placed before the receive statement.

Most important things are the rules to store the transferred messages in order to ensure that all in-transit messages of available recovery lines are logged on stable storage. In MRT, message logging is based on the following three rules:

RULE 1. *Messages sent from $I_{q,i}$ ($i \geq 0$) to $I_{p,j}$ with $j > i$ must be logged.*

RULE 2. *All messages sent from $I_{q,i-1}$ to $I_{p,i-1}$ ($i \geq 1$) are not logged iff*
(1) $C_{p,i} \rightarrow C_{q,i+1}$, and
(2) $(\forall s(s \neq p, q))(C_{q,i} \rightarrow C_{s,i+1}) \Rightarrow (C_{p,i} \rightarrow C_{s,i+1})$

RULE 3. *A message from $I_{q,i}$ to $I_{p,i}$ ($i \geq 0$) must be logged if the time elapsed since $C_{q,i}$ exceeds the time elapsed since $C_{p,i}$.*

Examples of the three logging rules are shown in Fig. 6.1. Message m_1 from $I_{3,0}$ to $I_{2,1}$ must be stored due to Rule 1. Message m_2 from $I_{2,1}$ to $I_{1,1}$ is not logged due to Rule 2, where $(C_{1,2} \rightarrow C_{2,3}) \wedge ((C_{2,2} \rightarrow C_{3,3}) \Rightarrow (C_{1,2} \rightarrow C_{3,3}))$. But message m_3 from $I_{1,1}$ to $I_{0,1}$ should be logged based on Rule 2 since $\neg((C_{1,2} \rightarrow C_{3,3}) \Rightarrow$

[2]This is a logging rule of ROS explained in Section 6.
[3]In ROS, processes with the same smallest checkpoint index never roll back.

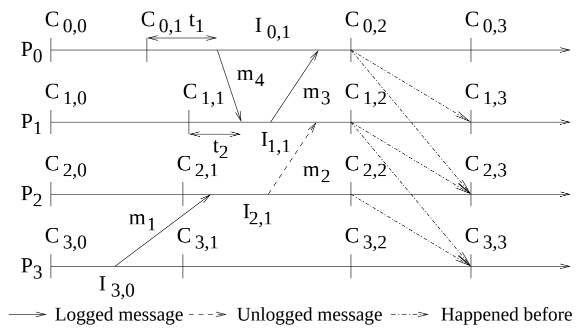


FIG. 6.1. *The rollback/replay distance.*

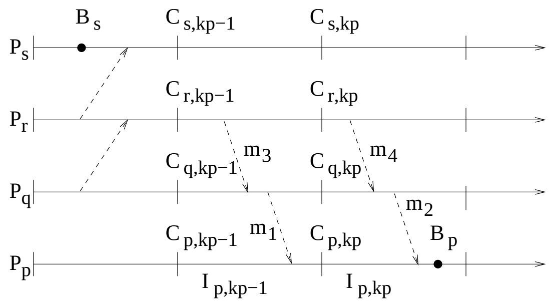


FIG. 6.2. *Dependence during replaying.*

$(C_{0,2} \to C_{3,3})$). Finally, message m_4 should be logged based on Rule 3 due to $t_1 > t_2$. To compare MRT with ROS, Rule 1 is kept, Rule 2 is modified, and Rule 3 is added.

Rule 1 allows recovery lines to be constructed at checkpoints with the same index because all in-transit messages of these global checkpoints are logged. It is used to avoid the domino effect [22], where cascading rollback propagation may force the system to restart from the initial state. Obviously, the most recent checkpoints of the distributed breakpoint can be used in the corresponding recovery line by using Shortcut Replay [29]. Unfortunately, the overhead is too high because every message may become an in-transit message of an available recovery line and thus it should be logged. The solution in both ROS and MRT is that each process could roll back one step during the recovery process. For instance, given an arbitrary distributed breakpoint $(B_0, B_1, \ldots, B_{n-1})$, in which the breakpoint B_i is placed in interval I_{i,k_i}, there always exists a corresponding recovery line $(C_0, C_1, \ldots, C_{n-1})$ where either $C_i = C_{i,k_i}$ or $C_i = C_{i,k_i-1}$. To satisfy both conditions that a small rollback distance and low message logging overhead, Rule 2 is developed. The three rules help to establish an upper bound for the replay time, which is described in Theorem 2.

THEOREM 2. *In MRT, there always exists a corresponding recovery line for any distributed breakpoint where*

- *The upper bound of the replay distance is* $2T$, *and*
- *The upper bound of the replay time is* $2T$.

Proof. The proof that the upper bound of the replay distance is $2T$ is similar to the proof for ROS [28]. Here we prove that the upper bound of the replay time is $2T$.

Consider a distributed breakpoint $(B_0, B_1, \ldots, B_{n-1})$ $(n \geq 2)$, the most recent global checkpoint

$$(C_{0,k_0}, C_{1,k_1}, \ldots, C_{n-1,k_{n-1}})$$

and the corresponding recovery line

$$(C_{0,h_0}, C_{1,h_1}, \ldots, C_{n-1,h_{n-1}})$$

in which $(h_i = k_i) \vee (h_i = k_i - 1)$ due to the upper bound $2T$ of the replay distance. We prove that the replay time from C_{p,h_p} to B_p in any process P_p is less than $2T$ by examining the worst case in which process P_p has to roll back one step, i.e. $h_p = k_p - 1$. The replay time may be long due to waiting for incoming messages, such as m_1 and m_2 in Fig. 6.2.

TABLE 7.1
Message logging overhead.

Programs	Number of processes	Execution time / checkpoint interval(sec)	Coordination time on each process (sec)	Total number of messages	Number of logged messages	Percentage
Message Exchange	4	19/2	0.002	120000	5507	4.59
	8	47/2	0.010	560000	15190	2.71
	16	77/2	0.559	1200000	27133	2.26
Poisson	4	23/2	0.004	149866	3760	2.51
	8	30/2	0.009	369084	6859	1.86
	16	59/2	0.101	864078	13356	1.55
FFT	4	18/2	0.027	270024	6373	2.36
	8	41/2	0.077	630056	9968	1.58
	16	95/2	0.884	1350120	18458	1.37
Jacobi Iteration	4	1802/2	8.380	49924	2411	4.83
	8	510/2	1.281	73848	3032	4.11
	16	268/2	2.120	153856	3442	2.24

The replay process of I_{p,k_p-1} depends on a set ψ of processes P_q that there exists either a direct message or a chain of messages (a causal path), e.g. m_3, m_1 in Fig. 6.2, from I_{q,k_p-1} to I_{p,k_p-1}, which is not logged. It is true that $((h_q = k_p-2) \vee (h_q = k_p-1)) \wedge (k_q \geq k_p-1) \wedge (C_{q,k_p-1} \rightarrow C_{p,k_p})$. In the case $(h_q = k_p-2) \wedge (k_q = k_p-1)$, there exists process $P_r (r \neq p, q)$ such that $(h_r \leq k_p-2) \wedge (k_r \leq k_p-1)$ and messages from I_{q,k_p-2} to I_{r,k_p-2} are not logged. If $k_r < k_p-1$, then $B_r \rightarrow B_p$ following Rule 2 (contradiction). If $(h_r = k_r-1) \wedge (k_r = k_p-1)$, then $C_{r,k_p-1} \rightarrow C_{p,k_p}$ (following Rule 2) and the recursive process is continued. The recursion is stopped at process P_s such that $((k_s < k_p-1) \wedge (C_{s,k_p-1} \rightarrow C_{p,k_p}))$ since the number of processes is finite. It also gives us $B_s \rightarrow B_p$ (contradiction). Therefore, all processes P_q in ψ have $h_q = k_p - 1$. Consequently, the replay time of I_{p,k_p-1} is only T due to Rule 3.

The replay process of I_{p,k_p} depends on a set ξ of processes P_s that there exists either a direct message or a chain of messages (a causal path), e.g. m_4, m_2 in Fig. 6.2, from I_{s,k_s} to I_{p,k_p}, which is not logged. These processes P_s satisfy $((h_s = k_p-1) \vee (h_s = k_p)) \wedge (k_s \geq k_p)$. In the case that all processes P_s have $h_s = k_p$, replay time of I_{p,k_p} is only T. In the others, some processes P_s have $(h_s = k_p-1) \wedge (k_s = k_p)$. Since replay time of I_{s,k_s-1} is only T (above proof), T could be added in the replay time of I_{p,k_p} due to waiting of replaying I_{s,k_s-1}. Therefore, the replay time of I_{p,k_p} is $2T$.

When I_{p,k_p-1} is replayed in T units time, other processes P_s in ξ, which has $(h_s = k_p-1) \wedge (k_s = k_p)$, are also replayed to C_{s,k_p} during T so that replaying I_{p,k_p} requires only T. Therefore, the replay time in process P_p from C_{p,h_p} to B_p is only $2T$. $\square$

7. Implementation. Checkpoints are taken periodically based on the defined checkpoint interval. However, checkpoints may be taken earlier when the incoming message is from a checkpoint interval with a higher index as shown above. This section describes the method to collect data to evaluate logging rules.

Due to the checkpoint interval index piggybacked on each transferred message, it is easy to decide which messages should be logged following Rule 1. To evaluate Rule 3, an additional value must be tagged on the transferred messages. Upon sending, the time t_1 elapsed since the last checkpoint is piggybacked on message m_4 as shown in Fig. 6.1. Due to $t_1 > t_2$, process P_1 will save m_4 to the trace files. If the incoming message does not satisfy both Rule 1 and Rule 3, it is kept in temporary storage in order to be evaluated later based on Rule 2.

When process P_p arrives at checkpoint event $C_{p,i+1}$, it will decide to store messages received in interval $I_{p,i-1}$ or not based on Rule 2. Of course, it requires tracking the happened-before relation between checkpoint events. This method uses the *knowledge matrix* as shown in [28]. However, both processes P_p at state $C_{p,i+1}$ and P_q at state $C_{q,i+1}$ cannot know accurately all processes P_s such that $C_{q,i} \rightarrow C_{s,i+1}$. Unfortunately, the upper bound of the replay time (and the rollback distance) in MRT will be larger than $2T$ if Rule 2 cannot be evaluated accurately. Only process P_s at state $C_{s,i+1}$ knows accurately which process P_q satisfies $C_{q,i} \rightarrow C_{s,i+1}$. Therefore, when each process P_p arrives at state $C_{p,i+1}$, it must broadcast the information of all processes P_r satisfying $C_{r,i} \rightarrow C_{p,i+1}$ to all other processes and receives the same information from them. Afterwards, process P_p can evaluate Rule 2 independently.

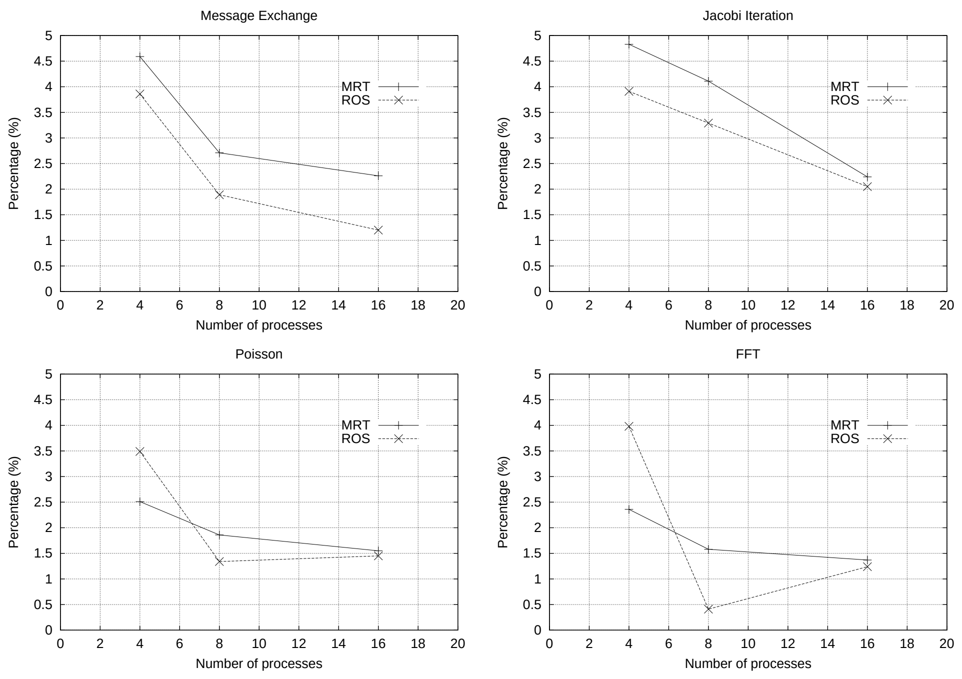


Fig. 7.1. *Percentage of the number of logged messages per the number of total messages.*

The efficiency of MRT is verified through several applications, such as Message Exchange, Jacobi Iteration, FFT and Poisson, shown in Table 7[4]. Message Exchange is a simple program in which messages are sent and received from one process to others. Poisson is a parallel solver for Poisson's equation. FFT performs a Fast Fourier Transformation. Jacobi Iteration is used to solve the system of linear equations. These results show that the number of logged messages is mostly less than 5% of the total number of messages. It is even better if the program's running time is longer and does not depend on the number of processes. A comparison of the efficiency between MRT and ROS is shown in Fig. 7.1. The ratios of the number of logged messages to the total number of messages in both methods are small and each can compare with the other one.

8. The four-phase-replay method. In order to know whether to store the incoming messages for limiting the rollback distance during re-execution, the amount of monitoring must be increased. This means that more information about relations between events on processes must be obtained. Consequently, the program's behavior may be influenced when the instrumented code slows down the execution of one or more processes. This effect is called the *probe effect* [18], which is a problem in cyclic debugging since the change of a program's behavior not only hides some errors shown in the target program's execution but also could show additional errors. Therefore, reducing the overhead of monitoring is important.

In MRT, monitoring consists of two main parts: (1) operations at send or receive events and (2) operations at checkpoint events. The first operations help to improve the knowledge for each process about the relations between events. In order to reduce this overhead, we have to reduce both the information piggybacked on transferred messages and operations at communication events. However, it must be ensured that enough information to evaluate the three logging rules in MRT is provided. In order to collect the happened-before relation between checkpoint events on the fly, the piggybacked data on transferred messages as shown in Section 7 have been optimized.

The second operation can be evaluated through operations at each checkpoint event. This work includes (1) collecting relation between checkpoint events by a coordination among processes (See Section 7), (2) processing data and logging messages (based on Rule 2), and (3) taking a checkpoint. These are the main reasons that cause the delay in program's execution. The problem (3) (taking a checkpoint) can be handled by means of

[4]The coordination time on each process is the time that the process waits to receive information of $C_{q,i} \rightarrow C_{r,i+1}$ and $C_{p,i} \rightarrow C_{r,i+1}$ from all other processes P_r.

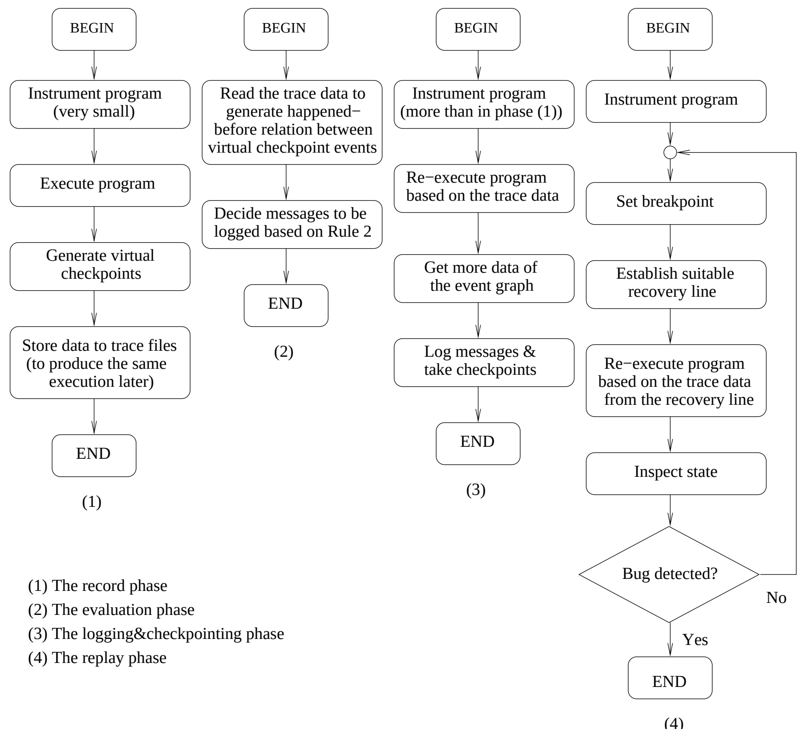

BEGIN
Instrument program (very small)
Execute program
Generate virtual checkpoints
Store data to trace files (to produce the same execution later)
END
(1)
BEGIN
Read the trace data to generate happened−before relation between virtual checkpoint events
Decide messages to be logged based on Rule 2
END
(2)
BEGIN
Instrument program (more than in phase (1))
Re−execute program based on the trace data
Get more data of the event graph
Log messages & take checkpoints
END
(3)
BEGIN
Instrument program
Set breakpoint
Establish suitable recovery line
Re−execute program based on the trace data from the recovery line
Inspect state
Bug detected?
No
Yes
END
(4)
(1) The record phase
(2) The evaluation phase
(3) The logging&checkpointing phase
(4) The replay phase

Fig. 8.1. *The four-phase-replay method.*

incremental checkpointing [6] and *forked checkpointing* [21]. By using forked checkpointing, a child process is created and it will take a checkpoint while the main process continues its execution. In addition, only data modified in comparison with the previous checkpoint are stored in incremental checkpointing. But problems (1) and (2) cannot be solved in MRT on the fly.

To avoid the probe effect, Leu and Schiper [16], and later Teodorescu and Chassin de Kergommeaux [27] introduced a new solution, in which only minimum tracing data are required to allow re-execution of programs. Thus the initial execution is assumed to be only slightly perturbed and afterwards the replayed executions are used to collect more information. This is similar to incremental tracing [3, 12]. This idea can be applied to avoid the probe effect when using MRT.

The record&replay mechanisms with two phases are unsuitable in this case. Therefore, the *four-phase-replay* method is introduced. As shown in Fig. 8.1, it has four steps as following:

1. *The record phase*: It requires only to collect the correct event occurrence timings in the initial execution since it is able to replay the program with clock synchronization algorithms [24]. The checkpoint events are also created in the initial execution but no checkpoint images are actually stored. Such checkpoint events are called *virtual checkpoints*. It is expected that the target program's behavior is affected very slightly due to the small instrumentation in this step.

2. *The evaluation phase*: The second step is to produce the necessary information used for message logging rules in MRT. Fortunately, the direct happened-before relation between pairs of events, e.g. send and corresponding receive events of a message, is enough to exhibit the happened-before relation between virtual checkpoint events by processing off line. Hence, Rule 2 can be evaluated based on these data.

3. *The logging&checkpointing phase*: After executing the evaluation phase, a re-execution is required to store in-transit messages and take real checkpoints. Please note that all message logging rules in MRT can be evaluated in the logging&checkpointing phase. Of course, the re-execution in the logging&checkpointing phase will produce the same program behavior as shown in the initial execution by clock synchronization algorithms [24] and will not create serious effects.

4. *The replay phase*: When sufficient data are available, the program can be restarted at a suitable recovery line and the waiting time to arrive an arbitrary distributed breakpoint can thus be reduced during subsequent executions.

9. Discussion. The advantage of MRT compared to ROS is the small upper bound of the replay time.(The upper bound of the replay time of ROS is nT and of EROS is $3T$ [30].) The disadvantage of MRT is that it requires additional synchronization mechanisms to collect data. The synchronization could produce serious probe effects due to the delay of the target program's execution. However, the four-phase-replay method used for MRT, in which some logging rules are evaluated off line, has many advantages. First, the probe effect is avoided in this method. Monitoring can be increased in the logging&checkpointing phase to learn more about the program's internal states. Second, the information obtained by an off-line method is more adequate than an on-the-fly method and thus the decision of logging is more accurate and efficient. For instance, the on-the-fly logging method of ROS may log messages, which are useless in replay process [28]. The reason is that a process may not obtain accurate relations between checkpoints on the fly when it decides to log messages or not based on Rule 2. This problem is solved in the four-phase-replay method since relations between checkpoints can be evaluated based on trace data.

In the future, the four-phase-replay method used for MRT will be applied in Process Isolation [12], in which users can run and inspect only a group of processes. This technique requires a re-execution to store all messages coming from the outside processes. Therefore, this step can be integrated into the logging&checkpointing phase. The resulting technique allows to reduce both the waiting time and the number of processes during debugging long-running, large-scale parallel programs [13].

10. Conclusions. A major problem that prohibits cyclic debugging with breakpointing for parallel programs is the long waiting time. Although there are many efforts, the waiting time may still be rather long or the amount of trace data are too large with long-running, large-scale parallel programs. An approach to solving this problem is ROS, but its upper bound of the replay time depends on the number of processes.

The new method named MRT is shown in this paper. The upper bound of the replay time is independent of the number of processes and is only $2T$ at most. In addition, this method is really efficient in minimizing the number of logged messages. These characteristics are important to develop debugging tools for parallel programs, in which some degree of interactivity for the user's investigations is required.

The disadvantage of the MRT method is that it requires coordination among processes in order to get the necessary information used in message logging. This affects not only the autonomy of each process but also synchronization between them. Consequently, the four-phase-replay method is proposed. The four-phase-replay method has two advantages: (1) the process to evaluate logging rules is accurate and the message logging overhead is thus reduced, and (2) the probe effect is avoided. In addition, the four-phase-replay method can be integrated with Process Isolation in order to debug long-running, large-scale parallel programs.

Acknowledgments. We would like to thank our colleagues, especially Dieter Kranzlmüller, for discussions and their help to finish this paper.

REFERENCES

[1] K. M. Chandy and L. Lamport, *Distributed Snapshots: Determining Global States of Distributed Systems*, ACM Transactions on Computer Systems 3 (1985), pp. 63-75.

[2] J. Chassin de Kergommeaux, M. Ronsse and K. De Bosschere, *MPL*: Efficient Record/Replay of Nondeterminism Features of Message Passing Libraries*, In J. Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI99) (Sept. 1999), LNCS, Springer-Verlag, Vol. 1697, pp. 141-148.

[3] J. D. Choi, B. P. Miller, R. H. B.Netzer, *Techniques for Debugging Parallel Programs with Flowback Analysis*, ACM Transactions on Programming Languages and Systems (Oct. 1991), Vol. 13, No. 4, pp. 491-530.

[4] R. S. Curtis, and L. D. Wittie, *BugNet: A Debugging System for Parallel Programming Environments*, Proc. of the 3rd Intl. Conference on Distributed Computing Systems, Miami, FL, USA (Oct. 1982), pp. 394-399.

[5] E. N. Elnozahy, L. Alvisi, Y. M. Wang and D. B. Johnson, *A Survey of Rollback-Recovery Protocols in Message-Passing Systems*, ACM Computing Surveys (CSUR)(Sept. 2002), Vol. 34, Issue 3, pp. 375-408.

[6] S. I. Feldman and Ch. B. Brown, *Igor: A System for Program Debugging via Reversible Execution*, Proc. of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging (May 1988), University of Wisconsin, Madison, Wisconsin, USA, SIGPLAN Notices (Jan. 1989), Vol. 24, No. 1, pp. 112-123.

[7] J. Fowler and W. Zwaenepoel, *Causal Distributed Breakpoints*, Proc. of the 10th International Conference on Distributed Computing Systems (ICDCS) (1990), pp. 134-141.

[8] J. M. Hélary, A. Mostefaoui and M. Raynal, *Communication-Induced Determination of Consistent Snapshot*, IEEE Transaction on Parallel and Distributed Systems (Sept. 1999), Vol. 10, No. 9.

[9] P. Kacsuk, *Systematic Macrostep Debugging of Message Passing Parallel Programs*, Distributed and Parallel Systems (DAPSYS'98), Future Generation Computer Systems, North-Holland (Apr. 2000), Vol. 16, No. 6, pp. 597-607.

[10] D. Kranzlmüller and J. Volkert, *NOPE: A Nondeterministic Program Evaluator*, Proc. of ACPC'99 (Feb. 1999), LNCS, Springer-Verlag, Vol. 1557, pp. 490-499.

[11] D. Kranzlmüller, *Event Graph Analysis for Debugging Massively Parallel Programs*, PhD Thesis, GUP Linz, Johannes Kepler University Linz, Austria (Sept. 2000), `http://www.gup.uni-linz.ac.at/~dk/thesis/thesis.php`.

[12] D. Kranzlmüller, *Incremental Tracing and Process Isolation for Debugging Parallel Programs*, Computers and Artificial Intelligence (2000), Vol. 19, pp. 569-585.

[13] D. Kranzlmüller, N. Thoai, and J. Volkert, *Error Detection in Large-Scale Parallel Programs with Long Runtimes*, Tools for Programs Development and Analysis (Best papers from two Technical Sessions, at ICCS2001, San Francisco, CA, USA, and ICCS2002, Amsterdam, The Netherlands), Future Generation Computer Systems (Jul. 2003), Vol. 19, No. 5, pp. 689-700.

[14] L. Lamport, *Time, Clocks, and the Ordering of Events in a Distributed System*, Communications of the ACM (Jul. 1978), Vol. 21, No. 7, pp. 558-565.

[15] T. J. LeBlanc and J. M. Mellor-Crummey, *Debugging Parallel Programs with Instant Replay*, IEEE Transactions on Computers (Apr. 1987), Vol. C-36, No. 4, pp. 471-481.

[16] E. Leu and A. Schiper, *Execution Replay: A Mechanism for Integrating a Visualization Tool with a Symbolic Debugger*, Proc. of CONPAR 92 - VAPP V, LNCS, Springer-Verlag (1992), Vol. 634.

[17] M. Mackey, *Program Replay in PVM*, Technical Report, Concurrent Computing Department, Hewlett-Packard Laboratories (May 1993).

[18] C. E. McDowell and D. P. Helmbold, *Debugging Concurrent Programs*, ACM Computing Surveys (Dec. 1989), Vol. 21, No. 4, pp. 593-622.

[19] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 1.1*, `http://www.mcs.anl.gov/mpi/` (Jun. 1995).

[20] R. H. B. Netzer and B. Miller,*Optimal Tracing and Replay for Debugging Message-Passing Parallel Programs*, Proc. of Supercomputing'92, Minneapolis, MN (Nov 1992), pp. 502-511. Reprinted in: The Journal of Supercomputing (1994), Vol. 8, No. 4, pp. 371-388.

[21] J. S. Plank, *An Overview of Checkpointing in Uniprocessor and Distributed Systems, Focusing on Implementation and Performance*, Technical Report UT-CS-97-372, University of Tennessee (Jul. 1997).

[22] B. Randel, *System Structure for Software Fault Tolerance*, IEEE Transactions on Software Engineering TSE (Jun. 1975), Vol. 1, No. 2, pp. 221-232.

[23] M. Ronsse, K. De Bosschere and J. Chassin de Kergommeaux, *Execution Replay and Debugging*, Proc. of the 4th International Workshop on Automated Debugging (AADEBUG2000) (Aug. 2000), pp. 5-18.

[24] Ch. Schaubschläger, D. Kranzlmüller and J. Volkert, *A Complete Monitor Overhead Removal Strategy*, Proc. of PDPTA '99, International Conference on Parallel And Distributed Processing Techniques and Applications, CSREA Press, Vol. 4, Las Vegas, Nevada, USA (Jun. 1999), pp. 1894-1897.

[25] D. F. Snelling and G. -R. Hoffmann, *A Comparative Study of Libraries for Parallel Processing*, Proc. of International Conference on Vector and Parallel Processors, Computational Science III, Parallel Computing (1988), Vol. 8 (1-3), pp. 255-266.

[26] R. Strom and S. Yemini, *Optimistic Recovery in Distributed Systems*, ACM Transactions on Computer Systems (Aug. 1985), Vol. 3, No. 3, pp 204-226.

[27] F. Teodorescu and J. Chassin de Kergommeaux, *On Correcting the Intrusion of Tracing Non-deterministic Programs by Software*, Proc. of EUROPAR'97 Parallel Processing, 3rd International Euro-Par Conference (Aug. 1997), LNCS, Springer-Verlag, Vol. 1300, pp. 94-101.

[28] N. Thoai, D. Kranzlmüller and J. Volkert, *ROS: The Rollback-One-Step Method to Minimize the Waiting Time during Debugging Long-Running Parallel Programs*, High Performance Computing for Computational Science, selected Papers and Invited Talks of VECPAR 2002, LNCS, Springer-Verlag, Vol. 2565, Chapter 8, pp. 664-678.

[29] N. Thoai and J. Volkert, *Shortcut Replay: A Replay Technique for Debugging Long-Runnining Parallel Programs*, Proc. of the Asian Computing Science Conference (ASIAN'02), Hanoi, Vietnam (Dec. 2002), LNCS, Springer-Verlag, Vol. 2550, pp. 34-36.

[30] N. Thoai, D. Kranzlmüller and J. Volkert, *EROS: An Efficient Method for Minimizing the Replay Time based on the Replay Dependence Relation*, Proc. of the 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing (EUROMICRO-PDP 2003), IEEE Computer Society, ISBN 0-7695-1875-3, Genova, Italy (Feb. 2003), pp. 23-30.

[31] F. Zambonelli and R. H. B. Netzer, *Deadlock-Free Incremental Replay of Message-Passing Programs*, Journal of Parallel and Distributed Computing 61 (2001), pp. 667-678.

Edited by: Zsolt Nemeth, Dieter Kranzlmuller, Peter Kacsuk
Received: April 4, 2003
Accepted: June 5, 2003

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 13–22. http://www.scpe.org

ISSN 1895-1767
© 2005 SWPS

APPLICATION OF P-GRADE DEVELOPMENT ENVIRONMENT IN METEOROLOGY

RÓBERT LOVAS[‡], PÉTER KACSUK[‡], ÁKOS HORVÁTH[¶] AND ANDRÁS HORÁNYI[¶]

Abstract. The main objective of a meteorological nowcasting system is to analyse and predict in ultra-short range those weather phenomena, which might be dangerous for life and property. The Hungarian Meteorological Service developed a nowcasting system (MEANDER), and its most computational intensive calculations have been parallelised by means of P-GRADE graphical programming environment. In this paper the application of P-GRADE environment is demonstrated in a real-size meteorological problem. We give an overview on the parallelisation of MEANDER system applying P-GRADE environment at the different stages of parallel program development cycle; specification, design, debugging and performance analysis. However, the paper focuses on a novel approach of parallel debugging, which combines ideas from the field of formal methods and model checking techniques with advanced parallel debugging methods in a user-friendly way.

Key words. Parallel programming, graphical development environment, numerical weather prediction, debugging, formal methods, performance analysis

1. Introduction. MEsoscale Analysis Nowcasting and DEcision Routines, MEANDER [1][2] developed by the Hungarian Meteorological Service (HMS) has a crucial task in the protection of life and property. Taking into account all the available meteorological observations the numerical weather prediction system can help the meteorological service to issue weather warnings, which are essential for storm warning at Lake Balaton, for aviation, and for other industrial partners. MEANDER system consists of several computational intensive procedures that cannot be executed in time without efficient parallelisation.

On the other hand, MTA SZTAKI and University of Miskolc developed a graphical programming environment, called P-GRADE [3][4] that is able to support the entire life-cycle of parallel program development. In a joint project of HMS and MTA SZTAKI, we applied the P-GRADE environment in order to parallelise the complex sequential FORTRAN and C/C++ code of MEANDER system.

In this paper we introduce briefly the fundamentals of MEANDER system (see Section 2) and P-GRADE programming environment (see Section 3) as well as the way of parallelisation over the design (see Section 4), the debugging (see Section 5) and the performance analysis (see Section 6) phases of program development. However, the paper focuses on a novel approach of parallel debugging (see Section 5), which combines ideas from the field of formal methods and model checking techniques with advanced parallel debugging methods in a user-friendly way. Furthermore, our experiences concerning the usage of P-GRADE, and some comparative performance measurements involving supercomputer and cluster environments, are also presented to illustrate the efficient applicability of P-GRADE environment on both platforms. Finally, we summarise the related works (see Section 7), and our achievements (Section 8).

2. MEANDER nowcasting system. The main objective of a meteorological nowcasting system is to analyse and predict in the ultra short-range (up to 6 hours) those weather phenomena, which might be dangerous for life and property. Typically such events are snowstorms, freezing rain, fog, convective storms, wind gusts, hail storms and flash floods. MEANDER nowcasting system [1][2] of the Hungarian Meteorological Service is not only capable to predict those severe weather events but, if needed, it is also able to issue automated warnings.

For the development of MEANDER nowcasting system the following main meteorological observations could be used as input data: (i) Hourly surface measurements are available on about a 40 km horizontal coverage. (ii) Some vertical soundings of the atmosphere (their number and availability is rather limited in space and time). (iii) Radar data are available at every 15 minutes using different radars covering the whole Carpathian Basin. (iv) Satellite images on infrared, water vapour and visible channels. (v) Lightning detection data over Hungary.

The development was constrained by few considerations, which were as follows: (i) The analysis and 3 hour forecasting should be available within *20 minutes* after the measurement time (this point is crucial in order to provide a timely analysis and prediction). (ii) The system should be able to provide "present weather" information for all the gridpoints of the domain of interest (the expression "present weather" means in meteorology

[‡]MTA SZTAKI Computer and Automation Research Institute, Hungarian Academy of Sciences
1518 Budapest, P.O. Box 63, {rlovas|kacsuk}@sztaki.hu
[¶]Hungarian Meteorological Service
1525 Budapest, P. O. Box 38, {horvath.a|horanyi.a}@met.hu

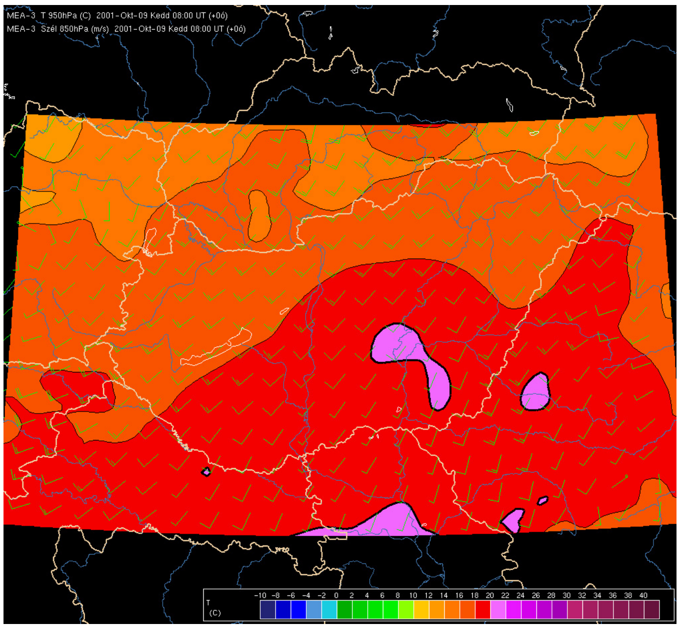

MEA-3 T 950hPa (C) 2001-Okt-09 Kedd 08:00 UT (+0ó)
MEA-3 Szél 850hPa (m/s) 2001-Okt-09 Kedd 08:00 UT (+0ó)
-10 -8 -6 -4 -2 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
T
(C)

Fig. 2.1. *Visualisation of interpolated basic parameters (wind, temperature)*

a special description of the main characteristics of the actual state of the atmosphere, like thunderstorm, fog, etc.). (iii) The system should be capable to provide automated warnings in case of specified weather phenomena (this aspect is an important one in order to prevent the consequences of the 'dangerous' weather).

Based on these constraints and considerations MEANDER system is going to be developed around three main segments: analysis (the determination of the present state of atmosphere), the linear segment (extra- and interpolation between the analysis fields and the forecasts of the numerical weather prediction models), dynamical modelling part (very high resolution numerical models with special physical parameterisation schemes for the precise prediction of weather events related to precipitation). The warning interface is smoothly fitted into the entire system. In the course of developments, until now, basically the first two segments are fully ready and operationally used, while the last one is under development and tests. Hereafter, briefly the analysis segment will be detailed, where the parallelisation of the computer code was developed.

The core of the analysis system is the so-called optimal interpolation scheme, where the basic meteorological variables are interpolated into the grid of the nowcasting system (basically the grid covers Hungary). The basic parameters are surface pressure, humidity, wind values, geodynamical height and temperature (see Fig. 2.1). To these basic parameters some others are added using radar and satellite information. From all the available information some derived quantities (these are typically type of cloudiness, visibility, phase of precipitation, some 'thunderstorm' parameters, moisture convergence, turbulence, convective available potential energy, etc.) are computed in order to get a deeper insight into the dynamics of the meteorological objects.

The parallelisation of the analysis segment can be based on the domain decomposition of model grid into subgrids (and the computations were carried out independently on the different subgrids) as it is described in Section 4.

On the other hand, the parallel execution of different calculations of basic and derived quantities (i.e. functional decomposition) can be a feasible way of parallelisation if there are enough computational resources.

3. P-GRADE development environment. The central idea of P-GRADE system [3][4] is to support each stage of parallel program development life-cycle by an integrated graphical environment where all the graphical views (applied at the different development stages) are related back to the original graph that defines the parallel program. The graph of parallel application is designed by the user but P-GRADE is responsible for generating all communication library calls automatically on the basis of the graphically described code. Since graphics hides all the low-level details of communication, P-GRADE is an ideal programming environment for average (or even beginner) developers who are not experienced in parallel programming, e.g. for chemists, biologists or meteorologists.

P-GRADE also supports the fast re-engineering of existing sequential applications by means of the applied hybrid graphical language, called GRAPNEL that is based on a hierarchical design concept supporting the bottom-up design methodologies beside the widespread top-down design concept. GRAPNEL provides predefined and scalable process communication templates that allow the user to describe the fundamental behaviour of the program with regular process topologies. In this way, the developer is able to construct complex and reliable applications in a relatively easy and fast way.

4. Design of parallelised MEANDER system. P-GRADE consists of three hierarchical design layers: (i) *Application Layer* is a graphical layer, which is used to define the component processes, their communication ports as well as their connecting communication channels. Shortly, the Application Layer serves for describing the interconnection topology of the component processes (see Fig. 4.1, Application window). (ii) *Process Layer* is also a graphical layer where different types of graphical blocks are applied: loop construct, conditional construct, sequential block, input/output activity block and graph block. The graphical blocks can be arranged in a flowchart-like graph to describe the internal structure (i.e. the behaviour) of individual processes (see Fig. 4.1, Process windows). (iii) *Text Layer* is used to define those parts of the program that are inherently sequential and hence only pure textual languages like C/C++ or FORTRAN can be applied at the lowest design level. These textual codes are defined inside the sequential blocks of the Process layer (see Fig. 4.1, at bottom of Process window labelled *Process: visib_s → visib_s_0*).

The usage of predefined and scalable process communication templates enables the user to generate complex parallel programs quickly. A communication template describes a group of processes, which have a pre-defined regular interconnection topology. P-GRADE provides such communication templates for the most common regular process topologies like Process Farm (see Fig. 4.1, Template window), Pipe, 2D Mesh and Tree, which are widely used among scientists. Ports of the member processes in a template are connected automatically based on the topology information.

In this project we mostly followed the domain decomposition concept in order to parallelise the six most time consuming modules of MEANDER, such as assimilation of incoming satellite images (see Fig. 4.1, 'satel_m' process and 'satel_s' Process Farm), processing of radar images (see Fig. 4.1, 'radar_m' process and 'radar_s' Process Farm), or calculation of derived quantities, i.e. the visibility (see Fig. 4.1, 'visib_m' process and 'visib_s' Process Farm).

Generally, we created one coordinator (master) process and one Process Farm template for each parallelised MEANDER routine. The 'master' process is responsible for dividing the model grid of MEANDER (over the Carpathian Basin) into subgrids as well as for the assignment of these partial tasks to 'worker' processes belonging to the Process Farm. When the worker received the 2D or 3D input data (via the communication channels and ports) to be processed on its part of grid, the worker invokes the corresponding FORTRAN or C/C++ sequential function from a sequential block (see Fig. 4.1, at the bottom of window labelled *Process: visib_s → visib_s_0*).

Preparing the parallelisation, the complex sequential functions of the original MEANDER source code were collected into separated object files. We rewrote (particularly parameterised) the sequential functions and also linked to the P-GRADE application. The simpler or shorter functions were inserted directly into the sequential blocks of Process Layer without changes.

Finally, the master process collects the partial results from the workers relying on the collective communication facilities of P-GRADE, and the master stores the collected information into a meteorological database in netCDF format.

As the Application Window depicts (see Fig. 4.1), the master processes of MEANDER calculations are connected to each other; they passes control messages to each other ensuring the appropriate scheduling among the calculation modules.

Each master process is also connected to a visualisation subsystem (*Ready + MetVis processes*), which is responsible for the qualitative on-the-fly visualisation of intermediate results on meteorological grids (see Fig. 4.1, *MetVis* window). Thus, the users are able to check the progress of calculation in run-time.

5. Debugging. After the successful design of MEANDER system, this section discusses in details the major issues as well as our novel solutions concerning the debugging phase of program development, since the reliability and the correctness are crucial criteria of MEANDER system.

During the debugging stage we took all the advantages of DIWIDE debugger [5], the built-in distributed debugger of P-GRADE environment. DIWIDE debugger provides the following fundamental facilities of par-

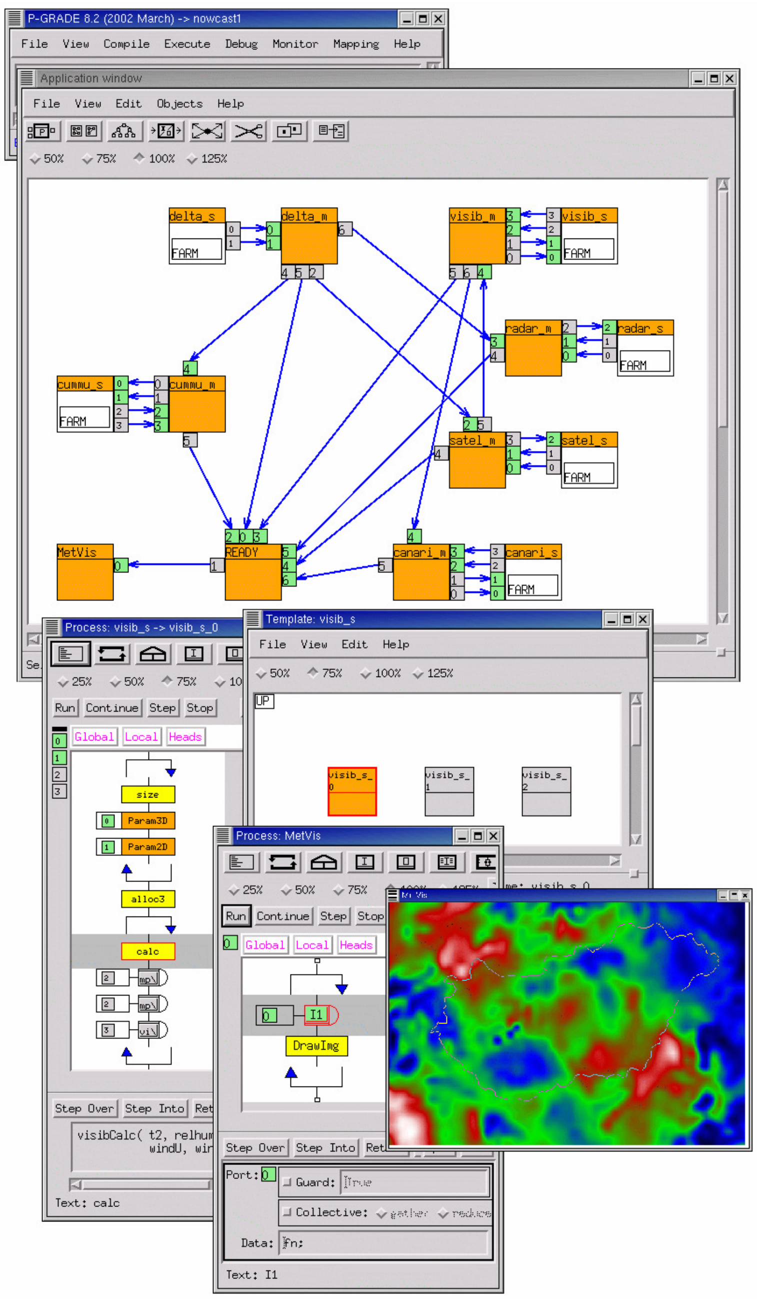


Fig. 4.1. *Parallelised MEANDER calculations in P-GRADE*

allel debugging; the step-by-step execution on both graphical and textual levels, graphical user interface for variable/stack inspection, and for individual controlling of processes.

Besides, the presented MEANDER system also served as a testbed for a novel parallel debugging methodology, which has been partially implemented in P-GRADE framework. This experimental debugging toolset tries to unify and also improve the research achievements of more completed projects [6][7][8]. The major aims of our debugging methodology can be summarised as follows:

- Handling the non-deterministic behaviour of P-GRADE programs.
- Application of formal methods and model checking techniques in parallel debugging, such as Petri-nets or temporal logic specifications.
- Reducing the needed user interactivity.

5.1. Temporal Logic Specification. As the first step of debugging, the user must specify the correctness properties (i.e. the expected program behaviour) of the developed GRAPNEL program using temporal logic (TL) formulas, since temporal logic has proved as an adequate framework for describing the dynamic behaviour of a system (program) consisting of multiple asynchronously executing components (processes) [11][12][17].

We defined the fundamental properties regarding the scheduler and the visualisation parts of the parallelised MEANDER system with three TL asserts. The first assertion (AssertLoad) expresses a basic resource requirement: the upper limit of the simultaneously working processes in Process Farms, which must be always fulfilled over the entire execution. The next assertion (AssertVisual) is used to express another property concerning the reliability of the visualisation subsystem; if any MEANDER routine has finished, its results must be eventually visualised in the future. Finally, the third assertion (AssertScheduling) formulates the expected execution order of MEANDER routines since the calculations have strict dependences to each other, and we also need proper scheduling to avoid the overload of available computational resources.

(1). *AssertLoad*: $[](\text{NumberOfActiveWorkers}() < 16)$
(2). *AssertVisual*: $(\forall x \in \mathbf{N} : 0 \leq x < 6)\ \text{CalcFinished}(x) \Rightarrow \Diamond\text{ResultVisualized}(x)$
(3). *AssertScheduling*: $\text{DeltaFinished}() \Rightarrow \Diamond(\text{CummuStarted}() \wedge \text{RadarStarted}() \wedge \text{SatelStarted}())$
$\wedge\ \text{SatelFinished}() \Rightarrow \Diamond\text{VisibStarted}() \wedge \text{VisibFinished}() \Rightarrow \Diamond\text{CanaryStarted}()$

The atomic predicates referenced from the above asserts, such as *NumberOfWorkers*, are implemented in dynamically loaded C libraries, and they use a simple predicate-DIWIDE interface [7] to obtain the actual state information of individual processes, e.g. actual values of variables, label of currently executed GRAPNEL block, or contents of already received messages.

5.2. Coloured Petri-net model. At the next stage of debugging, the formalism of coloured Petri-nets (CPN) [10] was chosen for expressing and composing a model for GRAPNEL programs because CPNs are well founded, have been widely used to specify parallel software systems and are supported by a number of tools for validation, simulation, analysis and verification.

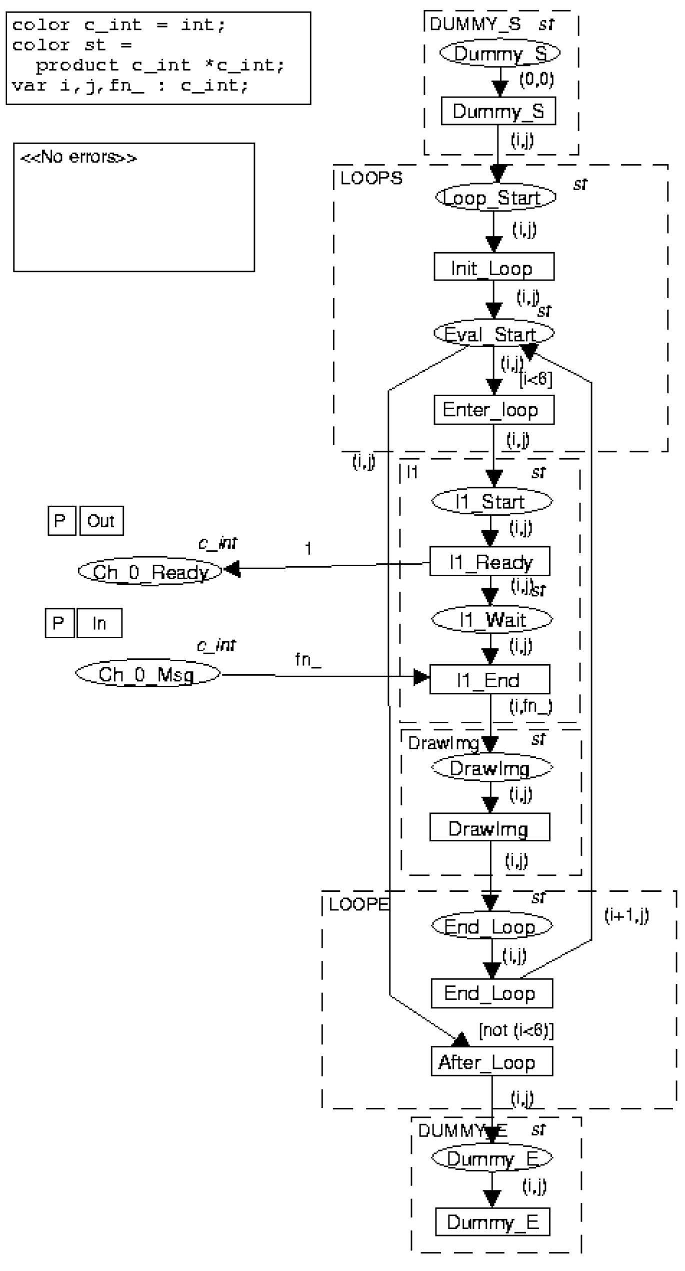
Some minor limitations and restrictions of GRAPNEL programs are required for the automated transformation to CPN model, e.g. at Process Layer the expressions of conditional and loop constructs must be dependent only on global control variables, which are declared in a separated section of GRAPNEL program. If a GRAPNEL program meets the given limitation rules, the new experimental GRP2CPN tool of P-GRADE system is able to generate a coloured Petri-net model from the developed GRAPNEL program for further simulation and analysis purposes.

The fundamental principles of the transformation from GRAPNEL programs to coloured Petri-nets are described in [6]. However, in our experimental debugging framework the place fusion approach was followed instead of the presented environmental place approach [6], and the positioning information of graphical items at Application and Process Layers are also exported into the CPN model in order to get topologically similar Petri-net to GRAPNEL program (compare the Process Metvis in Fig. 4.1, and its Petri-net model in Fig. 5.1). These modifications enable the user to identify easily the corresponding parts of GRAPNEL program and CPN model, and to simulate the behaviour of GRAPNEL programs from various aspects (liveness and home properties, deadlocks, etc.). The generated hierarchical CPN model is described in XML using the Document Type Description of Design/CPN tool [9] provided for importing and exporting models.

An example, Fig. 5.1 depicts a coloured Petri-net model generated from *MetVis* process of the parallelised MEANDER system. Basically, the presented Petri-net model has been constructed in a sub-page of our hierarchical model, and built up from the Petri-net patterns of FOR-type loop construct (denoted as 'LOOPS' and 'LOOPE'), input communication activity ('I1'), and sequential block ('DrawImg'). At the beginning of the model, a sequential block-like construct is located modelling the initialisation of global variables. In the left side of the Fig. 5.1, the end of the input communication channel/port is represented by two places (used for blocked message passing), and the global declaration of colours and tokens are placed.

Relying on the simulation facilities and the reachability analysis offered by Design/CPN tool the modelled parts of the nowcasting system has been verified successfully.

5.3. Improved macrostep-based execution. The general problem of constructing and replaying consistent global states of parallel/distributed systems must be also considered because the evaluation of erroneous situations depends on accurate observations. To solve these problems, Macrostep Engine (a part of DIWIDE dis-



```
color c_int = int;
color st =
  product c_int *c_int;
var i,j,fn_ : c_int;
```

```
<<No errors>>
```

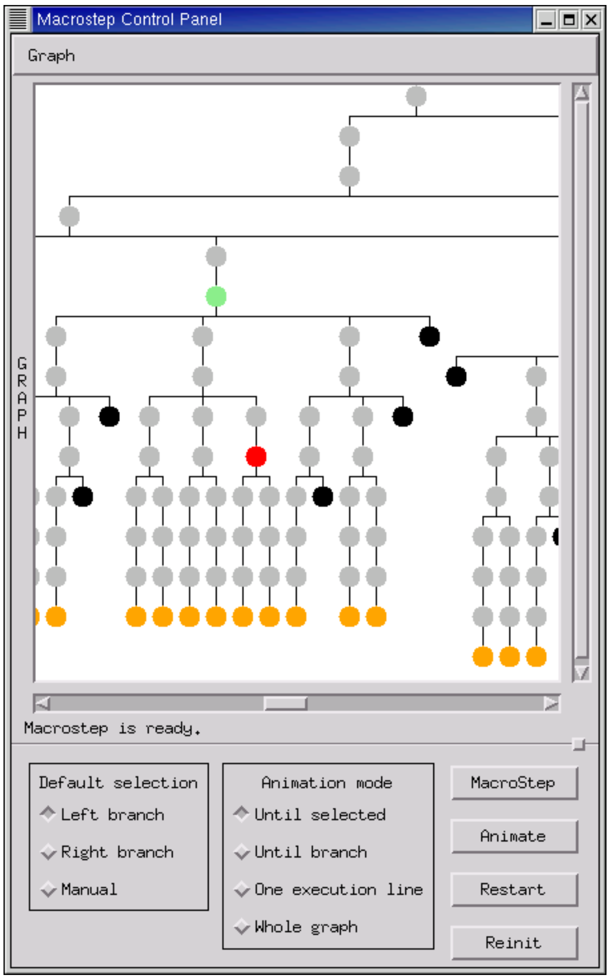
Fig. 5.1. *Coloured Petri-net model of MetVis process*

tributed debugger) provides strategies for the observation as well as replaying of consistent computation states relying on the so-called macrostep-by-macrostep execution. The idea of macrostep-based execution is based upon the concept of collective breakpoints[1] , which are placed on the interprocess communication primitives in each GRAPNEL process. The set of executed code regions between two consecutive collective breakpoints is called a macrostep. A detailed definition of macrostep concept is given in [8].

In this project we realised the analogy between the step-by-step execution mode of sequential programs (by local breakpoints) and the macrostep-by-macrostep execution mode of parallel programs.

The Execution Tree [7][8] (see Fig. 5.2) is a generalisation of a single execution path (based on macrosteps); it contains all the possible execution paths of a parallel program assuming that the non-determinism of the current

[1]A collective breakpoint is a set of local breakpoints, one for each process, that are all placed on communication instructions.



Macrostep is ready.

Default selection
- Left branch
- Right branch
- Manual

Animation mode
- Until selected
- Until branch
- One execution line
- Whole graph

Fig. 5.2. *Macrostep Control Panel*

program is inherited from wildcard message passing communications. In idealistic case, the task of Macrostep Engine would be the exhaustive traversing of the complete Execution Tree. In fact, when a developer tries to debug a real-size program systematically, the user has to face the problem of combinatorial explosion of possible execution paths. Another drawback of the original macrostep concept is that Macrostep Engine is able to detect only a few types of critical failures itself, such as deadlocks or wrong terminations.

In order to improve the macrostep concept, Macrostep Engine is able to cooperate with an external Temporal Logic Checker [7] (TLC) Engine comparing the user-defined TL specification (see Section 5.1) to the observed program behaviour macrostep-by-macrostep. Hence, the debugger framework has new capabilities to detect a wider spectrum of programming/implementation errors, and to cut the branches of Execution Tree earlier than the original macrostep concept.

Meanwhile, a CPN analyser tool[2] can be applied for simulating the program execution from some critical aspects (synchronisation, number and ordering of passed messages, termination, etc.) by means of the automatically generated CPN model (see Section 5.2). The simulation (i.e. the construction of Occurrence Graph [9] of Petri-net model) and the analysis of the results are faster by magnitudes of orders than the real program execution since most of the user's sequential code are not involved in the model. Based on the discovered failures the developer can steer the verification (i.e. traversing of Execution Tree by macrosteps) towards the found suspicious situations.

If an erroneous situation is detected, DIWIDE enables the user to inspect either the global state of GRAP-

[2]Actually we used Design/CPN v4.0.2.

NEL application or the state of any individual process. Depending on the situation, the programming bug can be fixed using the graphical editor of P-GRADE, or the entire application can be replayed to get closer to the origin of the detected erroneous situation.

By the combined efforts of DIWIDE distributed debugger, Macrostep Engine, Temporal Logic Checker, and coloured Petri-net generator/analyser we eliminated several critical programming bugs from the parallel version of MEANDER system.

6. Performance analysis. According to the available computational resources the actual number of worker processes in each scalable Process Farm can be set by a dialog window in P-GRADE. Currently, the Hungarian Meteorological Service works with an SGI Origin 2000 server equipped by 16 MIPS R12000/450MHz processors, and MTA SZTAKI installed a self-made Linux cluster containing 29 dual-processor nodes (Pentium III/500MHz) connected via Fast Ethernet. The parallel version of MEANDER system has been tested on both platforms. Based on our measurements the execution times of individual parallelised calculations depend particularly on the floating-point calculation performance of applied CPUs (assuming the same number of involved CPUs). To take an example, the visibility calculation was executed on the SGI supercomputer within 43 sec (see Fig. 6.1, in the middle of PROVE window), and it took approximately 116 sec to calculate on MTA SZTAKI's cluster. The rate between the execution speeds is almost equal to the reciprocals of the processors' SPECfp rates due to the computational (and not communication) intensive feature of the visibility calculation as PROVE performance visualisation tool [4] depicts. Some calculations, such as the processing of satellite images, were executed on the same time on both platforms; but the processing of radar data was slower on the cluster by a magnitude of order due to the long initialisation time (sending radar images) and the relatively short processing time.

In details, PROVE, as a built-in tool of P-GRADE system, can visualise either event trace data, i.e. message sending/receiving operations, start/end of graphical blocks in a space-time diagram (see Fig. 6.1, Prove window), or statistical information of the application behaviour (see Fig. 6.1, Process State and Execution Time Statistics Windows). In all the diagrams of PROVE tool, the black colour represents the sequential calculations, and two different colours used for marking the sending and the receiving of messages. The arcs between the process bars show the message passing between the processes.

Hereby, PROVE assisted us to focus on performance bottlenecks and to fine-tune of our application. On HMS's 16-processsor SGI computer the parallelised MEANDER calculations are executed within 2 minutes (ca. 15 times faster than the sequential version on an HP workstation). By the end of the joint project, almost the same performance results were achieved on MTA SZTAKI's cluster utilising all the 58 processors, the difference between the execution times was less than 5%. However, we had to take into consideration the saturation of speedup curves by each calculation as well as the limited size of operational memory offered by the cluster when the sizes of Process Farms were increased on the cluster.

Finally, a specialised visualisation tool, HAWK (Hungarian Advanced Workstation) developed by the Hungarian Meteorological Service provides a detailed and quantitative visualisation of the calculated meteorological parameters for the specialists and for other external partners (see Fig. 2.1, temperature and wind at 850 hPa on the basic grid of MEANDER).

7. Related works. Over the past few years P-GRADE system has been installed on several high performance clusters and supercomputers in the frame of various research projects and co-operations. Beside the presented meteorological application, P-GRADE is applied by physicist, chemists and students for designing parallel applications, and two other real-size applications were developed and demonstrated on scientific forums; a simulator for collision of galaxies (N-body problem), and an urban traffic simulator. Relying on the achievements of the presented work a national project has been launched for supporting complex collaborative work and its application for air pollution forecasting (elaboration of smog alarm plans) by means of P-GRADE environment and GRID infrastructure [16].

To provide more efficient support for long-running parallel applications on clusters, MTA SZTAKI have developed new experimental tools in P-GRADE framework with facilities for distributed checkpointing, process migration, dynamic load balancing, fault-tolerance execution and interfacing to job schedulers.

This paper focuses mainly on the debugging issues (see Section 5) of parallel applications following the active control approach, similarly to other state-of-the-art debugging frameworks [13][14][15][17]. The major advantages of the developed system comparing to the related works can be summarized as follows.

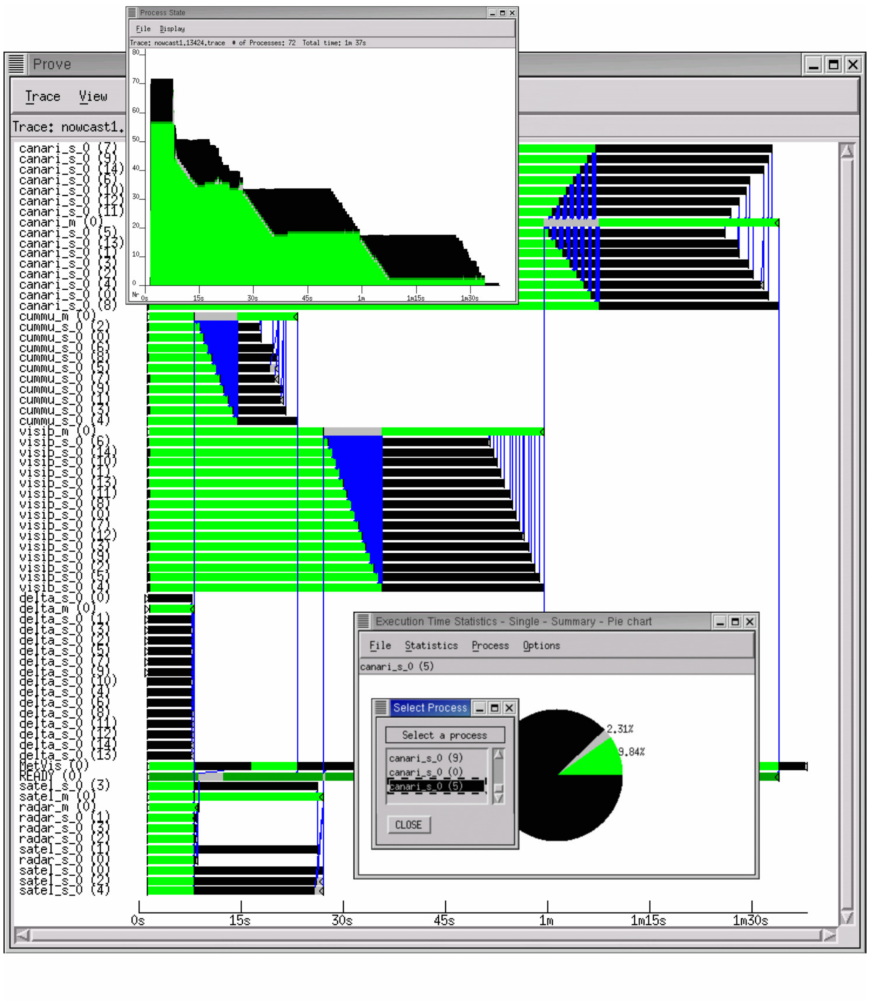


Fig. 6.1. *Performance visualisation with PROVE on SGI*

- Application of temporal logic assertions (instead of traditional predicates) offers more powerful expressiveness for the users to specify correctness criteria, and the universe, where the assertions are evaluated, is the result of real program execution (and not modeling).
- Automatic generation of coloured Petri-net model based on the GRAPNEL program code, which can assist to the user in steering of debugging activity.
- Macrostep-based active control enables the user to debug the parallel application similarly to the step-by-step execution of sequential programs.

However, our experimental debugging framework is strongly tightened to the GRAPNEL graphical language thus, it cannot be applied directly for legacy parallel application contrary to MAD environment [13] or STEPS/DEIPA [14].

8. Summary. As the presented work demonstrates, P-GRADE environment provides easy-to-use solutions for parallel program development even for non-specialist programmers at each stage of software development life-cycle. Hungarian Meteorological Service parallelised successfully the most time consuming parts of its model for ultra-short range weather forecast with P-GRADE system. Hence, the execution time of the entire processing (from the measurement to the issuing of warnings) matches to the 20-minute time constraint in order to predict on time those weather phenomena which might be dangerous for life and property.

The parallel version of the ultra-short range weather prediction system has been demonstrated successfully at the 5th DataGRID conference involving the server of Hungarian Meteorological Service, the PC cluster of MTA SZTAKI, and some mobile devices placed on the site of the conference. We would like to thank József

Kovács and Márk Rajnai for their assistance with various aspects of the successful demonstration.

The presented nowcasting meteorological system provided valuable feedbacks for the further development of P-GRADE, particularly for the experimental debugging framework, which combines ideas from the field of formal methods and model checking techniques with advanced parallel debugging methods in a user-friendly way.

Acknowledgments. The work presented in this paper was supported by the Ministry of Education under Nos. OMFB–02307/2000, OMFB–02313/2000, the Hungarian Scientific Research Fund (OTKA) No. T042459 and IHM 4671/1/2003.

REFERENCES

[1] Á. Horváth: *Nowcasting System of the Hungarian Meteorological Service*, Fifth European Conference on Applications of Meteorology, OMSZ Budapest, 2001. pp. 13.

[2] I. Geresdi and Á. Horváth: *Nowcasting of precipitation type*, Idöjárás Vol. 104. No.4. 2000. December, pp. 241–252.

[3] P. Kacsuk, J. C. Cunha and S. C. Winter (editors): *Parallel Program Development for Cluster Computing: Methodology, Tools and Integrated Environments*, Nova Science Publ., New York, 2001

[4] P. Kacsuk, G. Dózsa, T. Fadgyas and R. Lovas: *GRADE: a Graphical Programming Environment for Multicomputers*, Journal of Computers and Artificial Intelligence, Slovak Academy of Sciences, Vol. 17, No. 5, 1998, pp. 417–427.

[5] J. Kovács, P. Kacsuk: *The DIWIDE Distributed Debugger on Windows NT and UNIX Platforms*, Distributed and Parallel Systems, From Instruction Parallelism to Cluster Computing, Editors.: P. Kacsuk and G. Kotsis, Cluwer Academic Publishers, pp. 3–12, 2000.

[6] Z. Tsiatsoulis, G. Dózsa, Y. Cotronis, P. Kacsuk: *Associating Composition of Petri Net Specifications with Application Designs in Grade*, Proc. of the Seventh Euromicro Workshop on Parallel and Distributed Processing, Funchal, Portugal, pp. 204–211, 1999.

[7] J. Kovács, G. Kusper, R. Lovas, W. Shreiner: *Integrating Temporal Assertions into a Parallel Debugger*, Proceedings of the 8^{th} International Euro-Par Conference, Paderborn, Germany, pp. 113–120., 2002

[8] P. Kacsuk: *Systematic Macrostep Debugging of Message Passing Parallel Programs*, Journal of Future Generation Computer Systems, Vol. 16, No. 6, pp. 609–624, 2000.

[9] K. Jensen: *Design/CPN Reference Manual*, Aarhus University-Metasoft,1996

[10] K. Jensen: *Coloured Petri Nets: A High Language for System Design Analysis*, Advances in Petri nets, LNCS 483, 342–416, Springer

[11] L. Lamport: *The Temporal Logic of Actions*, ACM Transactions on Programming Languages and Systems, 16(3):872–923, May 1994.

[12] Z. Manna and A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems - Specification*, Springer, Berlin, 1992.

[13] D. Kranzlmüller, S. Grabner, and J. Volkert: *Debugging with the MAD environment*, Environment and Tools for Parallel Scientific Computing (editors: J. Dongarra and B. Tourancheau), volume 23 of Parallel Computing, pp. 199–217. Elsevier, 1997.

[14] J. Lourenco, J. C. Cunha, H. Krawczyk, P. Kuzora, M. Neyman, and B. Wiszniewski: *An integrated testing and debugging environment for parallel and distributed programs* Proc. 23rd EUROMICRO Conference, pages 291–298, Budapest, Hungary, 1997. IEEE ComputerSociety.

[15] A. Tarafdar and V. K. Garg: *Predicate control for active debugging of distributed programs*, Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP-98), pages 763-769, Los Alamitos, March 30–April 3 1998. IEEE Computer Society.

[16] I. Foster and C. Kesselman: *Globus: a Metacomputing Infrastructure Toolkit*, International Journal of Supercomputing Application, May 1997.

[17] M. Frey and M. Oberhuber: *Testing and Debugging Parallel and Distributed Programs with Temporal Logic Specifications*, Proc. of Second Workshop on Parallel and Distributed Software Engeneering 1997, pages 62–72, Boston, May 1997. IEEE Coputer Society.

Edited by: Zsolt Nemeth, Dieter Kranzlmuller, Peter Kacsuk, Jens Volkert
Received: April 17, 2003
Accepted: June 5, 2003

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 23–33. http://www.scpe.org

ISSN 1895-1767
© 2005 SWPS

TOWARDS A ROBUST AND FAULT-TOLERANT MULTICAST DISCOVERY ARCHITECTURE FOR GLOBAL COMPUTING GRIDS

ZOLTAN JUHASZ*, ARPAD ANDICS†, KRISZTIAN KUNTNER† , AND SZABOLCS POTA†

Abstract. Global grid systems with potentially millions of services require a very effective and efficient service discovery/location mechanism. Current grid environments, due to their smaller size, rely mainly on centralised service directories. Large-scale systems need a decentralised service discovery system that operates reliably in a dynamic and error-prone environment. Work has been done in studying flat, decentralised discovery architectures. In this paper, we propose a hierarchical discovery architecture that provides a more scalable and efficient approach. We describe our design rationale for a k-ary tree-based fault-tolerant discovery architecture that also acts as an intelligent routing network for client requests. We show that it is possible to provide ad hoc multicast discovery of services even without end-to-end multicast availability. The system provides clients with the ability to search for services with incomplete information using descriptive service attributes. Our results demonstrate that this approach is well suited to wide-area discovery if fast discovery, high availability, scalability and good performance are crucial requirements.

Key words. Grid, global computing, discovery, multicast, hierarchical overlay, service

1. Introduction.

Service discovery is an essential component of global grid systems where the number of services can be in the order of millions. Services are dynamic; they may join or leave the system as they wish. Hardware or software components may fail at any time, services can be migrated to or service replicas can be launched at new locations. Consequently, without having an effective solution for locating suitable services in a wide-area environment, grids will not be usable.

A global discovery architecture must provide the user with the ability to discover services or resources dynamically, as in general it cannot be assumed that the exact machine name or URL of the service is known. To be highly available, it should also be fully distributed with no single point of failure. It should be scalable in order to cope with the increase in size, and it should provide powerful search facility to locate services based on incomplete information.

Different aspects of wide-area service location and discovery have been the focus of research in various contexts. New directory services such as X.500, LDAP, UDDI have been developed to facilitate attribute-based service/resource search. Extensions have been proposed to create wide-area name/directory systems and augment the DNS name service [3, 24]. These systems, however, still require knowledge of at least one directory service. Discovery has been proved to be an effective solution for this entry-point problem as demonstrated by several local area discovery frameworks [8, 11, 26]. It is also a key element of peer-to-peer (P2P) systems that, consequently, have proposed various discovery schemes [7, 15, 21, 6]. Overlay routing networks have been proposed in certain peer-to-peer and distributed multimedia systems mainly to provide better global properties and multicast traffic among non-multicast reachable system components [5, 7].

Although wide-area service discovery is crucial to grids, it has received somewhat less attention within the grid research community. Current grid environments typically rely on an LDAP-based directory service that has proven scalability, update and query problems [19]. Future grid infrastructures based on Web Services technology will rely on UDDI directories, but it is yet unclear how this will provide a solution for large-scale, wide-area systems.

In this paper, we propose and describe a hierarchical discovery architecture that integrates techniques developed in different areas of distributed systems. This architecture relies on infrastructure services to create a self-configuring hierarchical routing overlay network that provides effective support for global resource and service discovery; it provides multicast discovery without the need for universal multicast availability; it is scalable, fully decentralised, self-configuring and robust; can be updated dynamically. The system also supports the execution of complex search queries based on service interfaces and attributes and—acting as a gateway—provides seamless wide-area access to services and resources.

The structure of our paper is as follows. In Section 2 we briefly review recent results related to wide-area service location and discovery, focusing on discovery, hierarchical naming systems, overlay networks and multicasting. Section 3 outlines the main requirements for our proposed discovery architecture. Section 4

*Department of Information Systems, University of Veszprem, Egyetem u 10., Veszprem, 8200 Hungary. ({juhasz, kuntner, pota}@irt.vein.hu). Zoltan Juhasz is also with the Department of Computer Science, University of Exeter, UK.

†Computer and Automation Research Institute, Hungarian Academy of Sciences, Hungary. (andics@sztaki.hu).

concentrates on the design aspects of the system discussing the need for Broker services, flexible matching, and an overlay routing network. Section 5 focuses on the details of the configuration, operation and fault-tolerance of the hierarchical broker network. In Section 6 we describe our results to date obtained on small testbeds. We discuss potential ways for improving our system in Section 7, then end the paper with our conclusions.

2. Related work. With the widespread acceptance of the Internet and World Wide Web, the limitations of commonly used name and directory services have become apparent. Researchers have focused on various aspects of wide-area service discovery but little research has been done for global grid service discovery.

Accepted directory service technologies, such as LDAP [25] and UDDI [2] work on the assumptions that stored data is mainly static, updates are rare, and clients know how to connect to the directory. These systems do not provide global directory services and the removal of stale entries is a key problem.

Several alternative systems have been proposed as alternative wide-area naming and directory systems. Both the Intentional Naming System [3] and the Globe Location Service [24] use attribute-based matching and hierarchies to access information effectively. They however focus on information access rather than on providing access to services via discovery.

Several local discovery schemes and protocols have been proposed for network and directory discovery. Examples are Universal Plug-and-Play [23], Service Location Protocol [11], Salutation [22], Jini [26]. While they solve the discovery entry point problem, none of them provide a solution for wide-area discovery. The Service Discovery Service [8] architecture was an attempt to create a global discovery architecture. It is based on a hierarchical, secure network of directory nodes. Unfortunately, only plans existed for the wide-area support, no implementation and global tests have been reported. Another proposed hierarchical service discovery architecture is the Carmen system [17]. It is a dynamic, updatable wide-area service discovery scheme, but at the time of writing it lacks full wide-area support and security, and it has to be manually configured.

Another line of research is to use peer-to-peer computing [18] concepts for discovery. One example system is the scalable content-addressable network [20] that uses a d-dimensional hash-based routing overlay over the mesh of nodes. Another proposed P2P discovery system is by Iamnitchi and Foster for grids [14]. It is a decentralised, self-configuring architecture based on P2P routing in a flat network. Their system can use four different request-forwarding policies that are combinations of random and best neighbour routing with learning from experience. The Web Services Discovery Architecture [13] proposal is an interesting system, providing advanced query facilities. The architecture is based on Web Services technology, but uses P2P concepts to provide discovery. It is unclear however, how effective wide-area discovery is supported.

The main general problem with P2P discovery approaches is the unbounded discovery time due to the unbounded hops to travel during discovery, which makes their effective use in large discovery systems used by millions rather doubtful.

Internet content distribution systems rely on overlay networks, mainly to provide end-to-end multicast without router support. The Overcast [15] system was designed to work in a single source setting, building a distribution tree automatically and dynamically. Scattercast [7] on the other hand is designed to provide multicast between multiple sources and sinks. It relies on special infrastucture service agents, Scattercast Proxies, to make multicast available over a unicast network.

Although much work has been done on various aspects of wide-area service discovery to date, no discovery architecture yet exists that would provide seamless access to global services, offer a powerful and effective query functionality at the same time, and operate in a spontaneous, highly-available, fault-tolerant, and dynamic manner. Such a discovery system is necessary for the success of grids, and for this reason, it should be more in centre of attention of the grid research community.

3. Service discovery system requirements and design rationale. Designing a discovery architecture capable of supporting service and client populations in the order of hundred millions is not a trivial task. The system has to meet a set of demanding requirements. In the following, we only describe the most important ones.

The number one requirement for the discovery system is to provide seamless access to services at a global scale, since future grid users would like to access grid resources and services without any hassle, administrative difficulty or knowing that they are actually accessing the grid.

A good discovery system should support the discovery of services without *a priori* information about their location (*location transparency*), or about the number of instances a service may have in the network (*replication transparency*) using flexible and powerful associative search mechanisms. Since the discovered services most

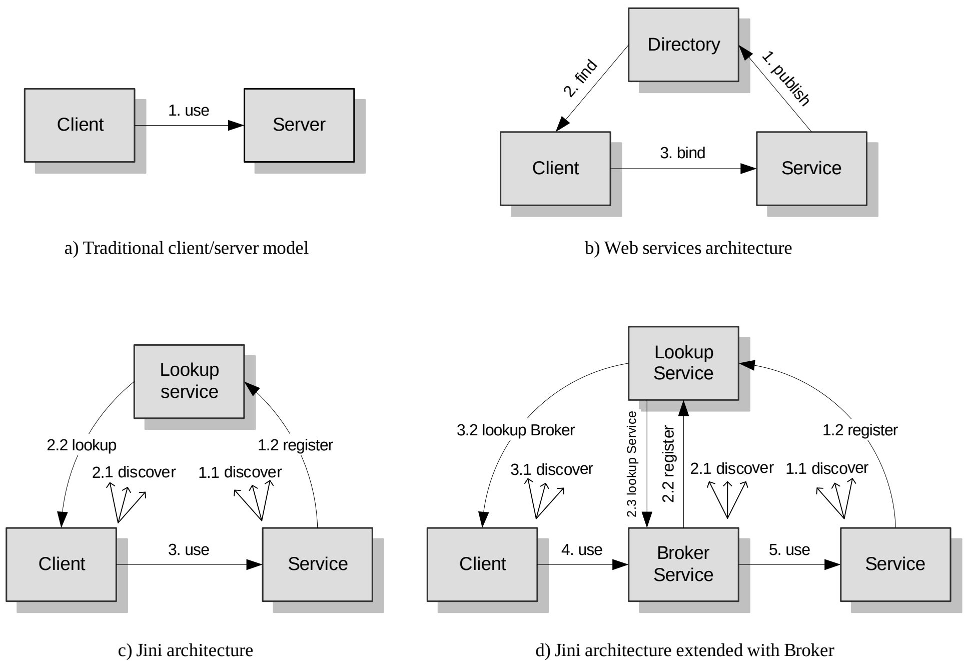


FIG. 2.1. *Different levels and approaches of de-coupling in client-server architectures.*

likely will be used by other programs, the search should be based on well-known service interfaces and additional service attributes. The system should provide means to perform complex queries effectively on a wide-area scale.

In addition, it should be possible to expand the system in size without changing either its overall structure or the internals of any of its components (*scalability*) due to either the increase of system components or to the increased load on certain components of the system (*performance transparency*). The system should also hide the location details of clients and services thus enabling the use of client and service mobility (*mobility transparency*).

The system should not become vulnerable to errors, it needs to be fault-resilient and robust. There cannot be a single point of failure in the system, and in the case of unrecoverable errors, the system should degrade gracefully both in its functionality and performance. In very large service communities, it is inevitable that services may come and go (leave willingly or due to a crash, and appear after initial start up or reboot), therefore the system should provide facilities for change management. It should be able to handle resources in a dynamic way and provide means to keep and present a soft system state the users.

4. Architectural design. In this section, we describe the general architecture of our discovery system step by step. First, we make a case for using Broker services, discuss how flexible search can be provided, then outline our proposed hierarchical, self-configuring and fault-tolerant discovery overlay network. To create this architecture we rely on the following techniques: multicast communication, distributed events, object mobility, proxy pattern, smart replication and caching.

The current architectural trend for distributed systems is based around the notion of directory services. Directories, as illustrated in Fig. 2.1(b), de-couple the traditionally tight client-server relationship, Fig. 2.1(a), by introducing a service registration, lookup and use operational mechanism. These techniques can be found in all major distributed technologies such as CORBA, Java RMI, and Web Services Architecture.

We believe that in their current form these directory services are not suitable for wide-area service discovery. As a search result, they all return a list of services to the client and require client interaction to select the suitable service. Furthermore, should a larger set of services to be discovered, the client will need to iteratively discover and look up several additional directories. Figures in Table 4.1 and 4.2 show that for a large number of directories, this can take considerable time.

Another obvious disadvantage of these systems is the so-called entry point problem; clients need to know the access details of at least one directory to be able to discover services. Should that central directory fail, the

TABLE 4.1
Unicast discovery times [msec] of lookup services running at different locations. The operation was initiated from Veszprem.

LUS location	Lookup service discovery time [msec]			
	min	median	mean	max
Local LAN	35	60	61	288
United Kingdom	140	165	195	9586
Norway	178	204	216	18999
USA	602	637	648	10914

TABLE 4.2
Minimum response time [msec] for lookup operations invoked from Veszprem on lookup services at different locations. Each lookup service was empty in order to obtain a lower bound on lookup execution time.

Location	Service lookup time [msec]			
	min	median	mean	max
Local LAN	4.8	5.2	5.5	29.1
Norway	54	58	69	10141
USA	154	160	174	8822

entire system becomes paralysed.

4.1. The Broker service. Jini extends this structure with multicast discovery, so the location of the directory no longer needs to be known. Once the client discovers the directory, as shown in Fig. 2.1(c), it can look up services then use the matching results the regular way. Multicast discovery removes the need for the client to discover directories iteratively. As the result of discovery is a set of directory proxy objects, caching can be used to keep directory references alive avoiding the need for initiating the discovery process repeatedly. This solution, however, still has problems. Firstly, it still relies on the client to carry out the potentially complex evaluation process of the returned search results. Secondly, caching directory proxies places extra memory and processing overhead on clients, which—in case of limited devices (PDAs, mobile phones)—is not permissible.

An intermediate service—a Broker—can be used to solve these problems. The Broker service (introduced in our Jini Metacomputer (JM) prototype [16], now called JGrid) is a mediator between clients and services, a single point of contact, whose role is to off-load work form the client. The main characteristic of the Broker is that it can perform tasks on behalf of the client and return only the result, not solely matching services, which distinguishes it from matchmaking [10] directory services, such as LDAP, UDDI, RMI registry or the Jini Lookup Service. This architecture, Fig. 2.1(d), allows the client to access services via a single contact point, without having to worry about the discovery and selection of services.

In the JM system, the Broker automatically discovers and caches Compute services. When clients ask the Broker to perform a computational task by sending a Java program to it, the Broker will try to find matching services stored in its lookup cache. The Jini event mechanism ensures that the service cache of the Broker will be automatically updated on changes to Compute service state. The Broker then selects the most suitable set of services, starts and controls the execution of the task and finally returns the result to the client.

4.2. Flexible matching. Besides providing service discovery, the other major role of the Broker is to select suitable services for clients. Normally, this would mean initiating several directory search operations and then evaluating the results. Our Broker provides more, however. To overcome the limitations of strict matching within the Jini lookup service (i.e. the only operator is '='), we provided search facility in the Broker with inexact matching capabilities. This is based on caching service proxies and attributes locally within the Broker service with our custom search operators ($<$, $\leq$, $>$, $\geq$; string matching; boolean AND, OR, NOT; set union $\cup$, intersection $\cap$) that facilitate the composition and evaluation of complex queries. Examples of possible queries are:

- *Return all available Compute services with more than 256 processors*,
- *Execute my task on a Compute service in Veszprem using 16 processors and minimum 1GB memory*, or
- *Return all services (any type) available in London and Vienna.*

With these extensions, the Broker can provide seamless access to services managed by brokers. The client can send a task for execution to the broker along with execution requirements, then wait for the result returned from the Broker upon its successful execution. Equally, the client can also use the Broker in the traditional matchmaking mode, only to receive a list of matching services, should it like to perform service selection itself, e.g. controlled by the user through a graphical interface.

4.3. The overlay routing network. The architecture we presented and discussed thus far is incapable of providing access to services outside the Broker's multicast region[1], hence only local services are available. In a practical wide-area system, such as grids, Brokers should provide seamless access to remote services as well. The two major technical issues to be solved are (*i*) the entry point problem (how to find the first point of contact), and (*ii*)how to provide access to remote services.

One approach, which would solve both problems, is to provide multicast routing on the Internet at a global scale. Theoretically, this would allow anyone to discover any service on the network assuming a large enough initial TTL value; consequently, our Broker could discover all interesting services. There are two problems with this approach though. Firstly, multicast is still not available everywhere on the network, and while large progress has been made to enable multicast on major backbones [9], it will take considerable time before (if ever) it becomes pervasive. Secondly, not being able to control generated network traffic and network use is a major obstacle. If clients use unnecessary large TTL values, their packets will reach those parts of the network, and consequently increase the load there, that are not intended recipients of these packets. Hence, the pure global multicast solution does not seem as a viable approach.

The other suggested approach is to federate Jini lookup services [1] in a peer-to-peer style by having lookup services register with other lookup services. In this way, a client could detect—without even using a Broker service—another lookup service by finding its proxy in the currently accessed lookup service. Using the proxy, the client can hop to the other lookup service and continue the search for suitable services. In essence, this is *hyperlinks* for Jini lookup services. One problem with this approach is that lookup services still need to know about each other to be able to register. If multicast is not available, either multicast packet tunnelling or name/address-based unicast discovery must be used, which requires manual administration and setting up. The other major problem is related to the use of the resulting lookup service structure. Clients need to be intelligent enough to detect lookup service proxies and to know how to perform hopping and subsequent search. This unnecessarily complicates client code. However, if we were to move this operation from clients to lookup services, the Jini lookup service specification would need to be changed. Finally, the resulting flat, peer-to-peer topology can guarantee neither the discovery of existing services nor the timely generation of search response. For these reasons we think that this approach is not viable either.

5. The broker hierarchy. In this paper, we propose a new approach to discovery: augmenting Jini lookup services with a hierarchical discovery overlay network that sits on top of lookup services. The aim of wide-area discovery systems is to make services *visible* to clients regardless of their location, which in our view does not require clients to physically discover and contact remote services by multicast packets. Our approach has several advantages.

- The introduction of the discovery overlay creates a layered architecture with a clear separation of responsibilities. The lowest, lookup service, layer is responsible for local service discovery and acts as an information source to the discovery layer (Brokers), whose role is to take client/application queries and perform wide-area service search.
- Large systems that rely on frequent global operations have to have a hierarchical structure. This simplifies system management, increases performance and efficiency, supports scalability, and localises traffic as well as faults.
- The system becomes open to future extensions of the discovery process without the need to change underlying services and directories. E.g., it will be possible to embed intelligent agents into Brokers that provide learning from experience, negotiation and similar high-level functionality.
- Unicast and multicast discovery can augment each other, resulting in an overlay network that can operate with and without multicast-enabled network segments.

We suggest a wide-area discovery architecture based on Broker services as shown in Fig. 5.1. Nodes (brokers) are interconnected to form a k-ary tree topology with optional connecting edges among nodes within each hierarchy level. In order to facilitate efficient search, we associate each layer with a given geographical unit, starting from continents down to organisational units. The reason for using a geographical hierarchy is that we

[1]The default multicast region is normally a single segment of a LAN. If network routers forward multicast packets, the time-to-live (TTL) value determines how many routers the packet will pass through; a packet with TTL = 5 will travel through 5 routers, hence network segments less than 5 router hops away from the source will receive the multicast packet.

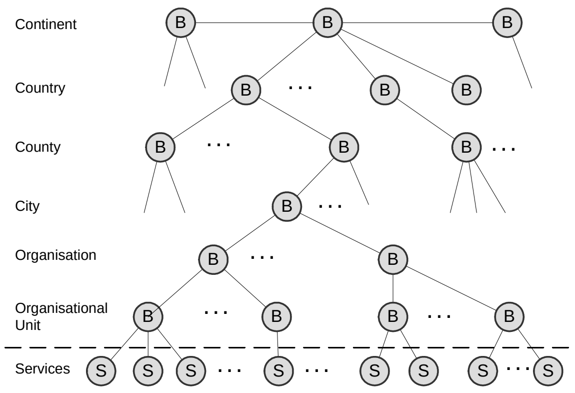


Fig. 5.1. *The high-level view of the discovery overlay system. Services are abstracted out and made available by Broker services connected in a hierarchy. A local (Organisational Unit) Broker can seamlessly provide access to distant services/resources for clients by contacting and querying other brokers.*

assumed in our system that services have rather fixed locations and institutional owners. Other hierarchies are possible as well, and we will return to this issue in Section 7.

5.1. Query routing. During operation, clients will contact and send a request to the lowest, organisational unit level broker. The request is given as a Jini `ServiceTemplate` object, which can specify a set of service interfaces and attributes a target service must match. A key attribute is `ServiceLocation` whose fields are identical to the layers of the broker hierarchy. By setting these fields to specific or 'don't care' `null` values, clients can indicate where the service they are looking for is, or similarly, where a submitted program should run.

Every service in the system, including brokers, uses its interface and a set of attributes to describe itself to the world. The `ServiceLocation` attribute is used to describe the location of the service during start-up. For services, this will determine which broker they connect to; for brokers, where in the hierarchy they have their place. Configuration is described in more detail in Section 5.2.

When executing tasks or requesting services, clients can pass the interface(s) of the required service(s) and descriptor attributes to the broker. If the client request does not specify the `ServiceLocation` attribute, the broker will search only its own services to find a match. Should `ServiceLocation` be specified, the broker first compares it with its own `ServiceLocation` attribute to see whether OrganisationalUnit fields match. If there is no match, the broker delegates the request to its parent for further search. The delegation process is shown in Fig. 5.2. Delegation is based on using proxies of parent and child brokers stored in local caches. Once a matching broker is found, an exhaustive subgraph search of its children is performed to find matching services. If, e.g., we specify `ServiceLocation` as Hungary and request 1000 processors, the discovery system will try to gather Compute Services totalling 1000 processors within the country, starting with local ones.

The overlay network provides very efficient routing. A query takes a maximum of $O(log_k S)$ hops, where k is the maximum degree of a broker and S is the number of services, to reach any service on the Internet. In our hierarchy, the maximum hop count is 11.

5.2. System configuration. In the previous sections we assumed the existence of the broker hierarchy. In this section we look at the process of configuring the discovery overlay system, focusing on how brokers will find and see each other.

Our primary requirement for system operation is minimal administration. We have designed the system to be self-configuring. Every node (Broker service) of the hierarchy performs multicast discovery to locate parent and child services in order to populate their internal broker proxy caches (Fig. 5.2). Each Broker has a designated level L_i, so each broker is only interested in finding L_{i+1} and L_{i-1} brokers. Leaf brokers naturally will not have broker children, they discover the exported services instead.

As discussed earlier, global multicast is not yet available, hence a multicast-only solution is not appropriate for configuring the system. Since Jini also offers unicast discovery, brokers can be configured to discover named lookup services (that hold broker proxies) using a set of URLs in the form of
`jini://name.of.lookup.service:port`. While it is normally assumed that clients can discover Broker services

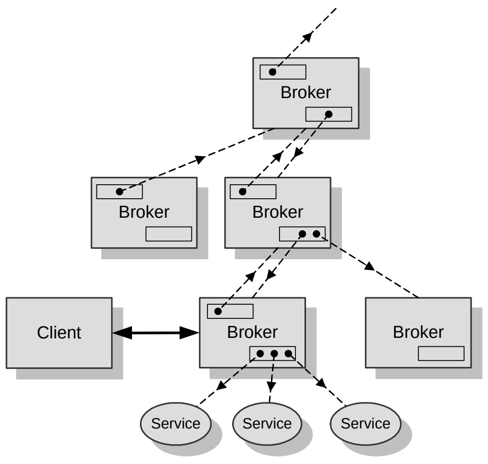


FIG. 5.2. *Request propagation in the discovery system. Client requests are routed based on attribute ServiceLocation.*

via multicast discovery on their local or institutional LAN, it is also possible to provide clients with a discovery URL to fall back to if multicast is not available. Most international and national backbones already (or soon will) provide multicast, therefore most likely it is the county/state-city-organisation layers where we need to rely on unicast discovery.

5.2.1. Multicast traffic. In a system where clients, services and the discovery system all rely to some extent on multicast, an important issue is the amount of generated multicast traffic and its effect on the network. In the following, we describe the multicast traffic patterns in our system and its effects, as well as ways to reduce the generated traffic.

The main source of multicast traffic for clients and services is discovery, during which multicast request packets are transmitted over the network. The maximum size of these packets is 512 bytes. The multicast request is sent during the client/service start-up sequence and repeated normally seven times with five-second delay intervals.

Lookup services also send multicast announcement packets to notify existing clients about their presence. These are sent by default in every two minutes. The packet size here is also max. 512 bytes. If all packets travel every segment of the Internet, the generated traffic would be high. For instance, ten million lookup services would generate a constant 40 MBytes/sec traffic by announcement packets only.

Multicast traffic can be reduced with localising multicast requests and announcements. Clients only need to discover leaf brokers that should be situated on a LAN of the client's home institution. For this, a small, typically < 5, TTL value can be used. Since brokers need to locate other brokers within small distances (few router hops away), small TTL values can be used within the broker hierarchy as well. The same is true for services, as they only need to locate lookup services within the institution. The correct threshold values depend on national network topologies, consequently require initial configuration. This approach results in mainly top-level brokers generating traffic on national and international backbones. Since their number is small (6 continents, 239 countries and approximately 3630 national regions [12]), the generated traffic is small.

Since there is no control over the initial TTL value of services and clients, two further techniques can be used to stop client and service-initiated multicast packets to travel too far. Routers can be set to stop multicast packets with an above threshold TTL, hence packets with too large TTL values, say 15, could be stopped. Another possible technique e.g. to stop multicast traffic leave a given country is to stop the Jini multicast port (4160) completely on the national gateway routers and only allow traffic over another port that tunnels the national broker multicast traffic to continental ones. These techniques could provide effective means in reducing international and national multicast traffic.

5.2.2. Unicast only mode. As discussed earlier, multicast discovery relies on sending request and announcement packets. Request packets originate from entities hoping to discover lookup services, announcement packets are sent by lookup services. Not being able to use multicast would imply loosing these functionalities, but our routing network ensures this will not happen. As long as the client can find one lookup service with a

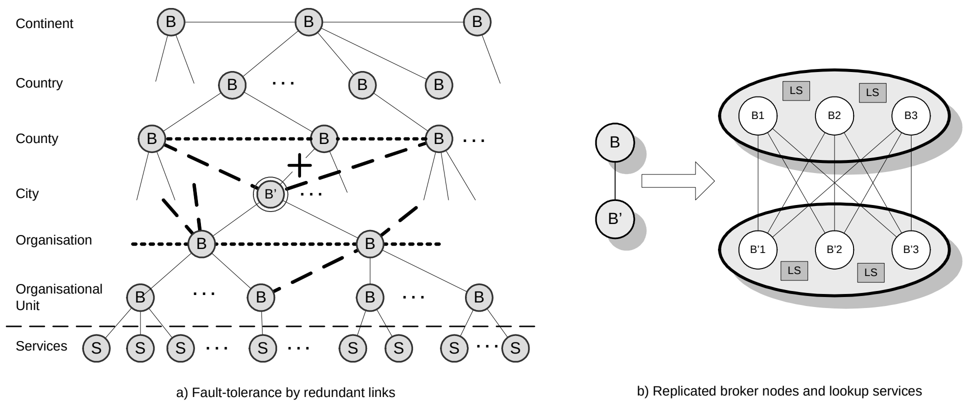


FIG. 5.3. *Link and node replication.*

leaf broker entry in it by unicast discovery, it can access any service on the network, as the broker is the only required contact point. The lack of announcement packets means that new lookup services will not be able to notify clients. Again, as long as the new lookup service can find at least one leaf broker, it will become part of the service network and clients will find the service automatically. The higher layers of the hierarchy can operate without any difficulty without multicast, as it is used only for configuration, provided the necessary configuration information is set-up.

5.3. Fault-tolerance. To guarantee service discovery, the overlay network needs to be fault-tolerant, robust and highly available. In this section we discuss what techniques can be used to achieve these goals.

The two major sources of faults in the overlay are network and node failures. To provide availability in the presence of these errors, multiple network paths and node replication can be used. In our base architecture, shown in Fig 5.1, nodes are connected in a tree topology by single links. To avoid communication breakdown among layers, redundant links can be introduced between a node and other parent layer nodes as shown in Fig. 5.3(a). Horizontal, peer links can also be introduced to provide an alternative route should all parent or children nodes fail.

Broker nodes can be made reliable by node replication. As shown in Fig. 5.3(b), a node in the overlay network can be viewed as a logical node that can have a replicated server implementation. Each broker node can be served by replicated Broker services that register with the same lookup service, hence will be automatically discovered by the relevant parent/child brokers. Note that replica brokers need not be placed at the same location; they can be physically distributed over the Internet, automatically providing alternative network paths to brokers. To avoid discovery failure, lookup services should be replicated as well, as also shown in Fig. 5.3(b). From the outside, the replica broker services are identical to the original brokers.

6. Results. To test our implementation we created small testbeds and carried out initial experiments to verify the operation and performance of the discovery architecture. We were especially interested in the response times of discovery queries on LANs and WANs, as well as the service matching time of brokers.

The crucial issue is how our discovery overlay performs with respect to the default Jini lookup service based discovery. The total lookup operation time in our overlay is $T_{lookup} = T_{delegation} + T_{matching} + T_{download}$, where $T_{delegation}$ is the time it takes for the query to reach its destination from the client, $T_{matching}$ is the time spent with service search in the leaf broker and $T_{download}$ is the time to download the service proxy objects of matching services to the client. Since $T_{download}$ is the same in both discovery systems, we ignore it in the comparisons.

6.1. Query routing performance. We performed experiments to measure delegation time, i.e. the response time of queries, to different layers in our overlay network. Two testbeds have been used. The first one, as shown in Fig. 6.1, consisted of 18 computers[2] connected to the same multicast-enabled LAN segment, while the second is a WAN configuration with computers located at two different locations in Hungary. We measured the query delegation time to different broker levels. The results of the first testbed are shown in Table 6.1. It can be seen that the average per hop time in the overlay network is around 3 *msec.*

[2]600 MHz P-III PCs running Windows 2000 and Java 1.4 HotSpot Virtual Machine.

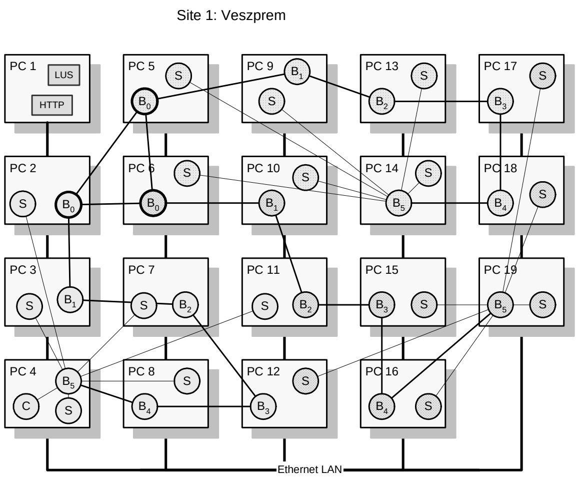


FIG. 6.1. *Wide-area search testbed. Services are assigned to three hypothetical geographical locations: America, Asia, Europe. Each computer hosts one Compute Service, S, and one node of the global broker hierarchy, B_i. Services are mapped to computers in a way to ensure all remote method invocations are performed via LAN, not in memory.*

TABLE 6.1
Minimum response time for lookup (discovery) operations initiated from Veszprem via the router overlay. No services were registered during these tests in order to obtain a lower bound on lookup execution time. Per hop time refers to the time necessary at the given level to forward the query to a parent or child node.

Delegation distance [hop]	Measured delegation time [msec]				Estimated time [msec]	
	average	min	max	per hop	total	per hop
1 (org unit)	4	0	30	1	4	2
2 (org)	6	0	50	2	6	3
3 (city)	8	0	40	2	9	5
4 (state)	10	0	40	2	14	5
5 (country)	12	10	40	2	24	10
6 (continent)	16	10	70	4	54	30
7 (other continent)	18	10	40	2	204	150
8 (country)	18	10	40	2	234	30
9 (state)	22	20	50	4	244	10
10 (city)	25	20	50	3	249	5
11 (org)	29	20	110	4	254	5
12 (org unit)	35	20	161	6	257	3

The second testbed was used to determine per hop time between two different WAN locations (9-14 *msec*). We also performed network latency measurement using `traceroute` to determine the typical latency figures for different parts of the internet. We analysed network traces between Europe, North and South America, Africa, Australia and Japan, and found that in national networks the latency is below 10 *msec*. Between countries the delay is in the order of 10 *msec*, while between continents it can be in the order of 100 *msec*. The estimated WAN per hop and accumulated response times are also given in Table 6.1. We estimate that with these broker services placed on backbone segments as infrastructure components, search results could be obtained within 100 *msec* in a continent and within few hundred milliseconds for truly global cases when intra-continent traffic is necessary.

6.2. Matching performance. To be able to reason about the total discovery time, we also measured matching time on one broker by creating several services with three attributes (`Processor`, `Storage`, `ServiceLocation`) altogether having 11 `String` and `int` fields. The measured per-service search times were 4.5 μsec for simple `int` matches (e.g. Number of processors=16), 10.6 μsec for compound `String` matches (e.g. Locality=Budapest OR Veszprem), and 17.3 μsec for compound `String` matches (e.g. (Locality=Shanghai AND MFlops $\geq$ 100000) OR organization = University of Veszprem). These results suggest that—assuming, say, maximum 1000 services per broker—matching time $T_{matching}$is in the order of 20 *msec*.

7. Discussion. Experiments with the current prototype system gave encouraging results. Without any optimisation and performance tuning, the overhead of one Broker is in the order of 2 *msec*, and global discovery can be performed in the order of hundred milliseconds. Note that these results are without using intermediate service caching. In comparison, the Globus system required seconds to discover resources of around 12 sites in case of a cache hit, and up to 10 minutes with empty GIIS caches[4].

We have also identified shortcomings of our system during the development. In our current scheme, each Broker service performs a parallel breadth-first search for services using its child Broker proxies. This, on one hand, generates large number of requests in the overlay network, on the other hand, the whole network needs to be searched if a non-existing service is requested. These problems call for a more intelligent query routing approach.

Another possible problem is the pre-determined hierarchy used in our overlay. It can be envisaged that other users or organisations would like to have services with different categorisation. Although it is certainly possible to embed other routing hierarchies in our system, that overlay would not be as efficient in its routing ability as the current one.

Finally, the geographical partitioning is not likely to be suitable for more than few million services, as the top layers of the hierarchy have low degrees. Assuming 6 continents, 239 Countries, 3630 regions, and 50027 cities [12] in the world, the average degree for continent, country and state level brokers is 47, 15 and 14, respectively. This would require many Broker services below city and organisational level, as well as result in a largely non-uniform distribution of services within the tree.

7.1. Future work. To address these issues and make our system more flexible, we are looking at ways to make the routing overlay nodes taxonomy-free, i.e. to create a general routing overlay without pre-defined classification.

This requires service attribute propagation and aggregation in the router overlay. Being able to provide content-based query routing will make search even more effective, as search request will only reach a broker if a potentially matching service can be found among its children. This architecture would naturally support the use of arbitrary service classifications and would facilitate the creation of user-defined service views. These modifications will also make it simpler to introduce redundant parent and peer links in the system. Since potential parent nodes do not belong to different “location” any longer, the fault-tolerant routing algorithm can be made simpler and more efficient.

Finally, in the next version of our overlay system, we intend to separate brokering and routing functionality by introducing router services for the overlay network, and broker services as interfaces (leaf nodes) between clients/services and the overlay network.

8. Conclusions. In this paper, we described a hierarchical discovery architecture that provides effective support for wide-area service discovery. The architecture is fully decentralised, self-configuring and scalable. It can be updated dynamically and acts as an intelligent router network for client queries. The hierarchy is based on broker services each performing multicast and unicast discovery.

The system supports ad hoc, attribute-based service discovery, even without end-to-end multicast network support and provides seamless wide-area access to services. Clients interact with the system by connecting to brokers at the organisational unit level that will delegate search requests to other nodes as necessary.

Our first experiments confirm that this approach is well suited to discovery in large grid environments. The discovery architecture has very good scalability, it is efficient for both local and global resource discovery.

Although the focus of our work and this paper is on computational grids, it is expected that future grid systems will provide access to a far wider range of services, varying from e.g. video-on-demand, context and position aware services to mobile entertainment, e-business and collaboration services. Users of these systems will require instantaneous results for search operations involving millions of services using incomplete information.

The overlay approach makes this fast search possible and effective. The system can scale with size without performance degradation and change in the underlying software architecture. For increased availability and reliability, replicated broker nodes can be added transparently.

Although in this current prototype we used a pure Java/Jini approach, our proposed system can be opened up to a wide range of service information data sources. With appropriate adaptors, the overlay network could read in information from alternative (LDAP, UDDI, etc.) directory services, creating a universal wide-area discovery architecture for all.

REFERENCES

[1] Federale Project. `http://federale.jini.org/`, 2003.

[2] Universal Description, Discovery and Integration (UDDI) Project. `http://www.uddi.org`, 2003.

[3] W. Adije-Winoto, E. Schwartx, H. Balakrishnan, and J. Lilley. The Design and Implementation of and Intentional Naming System. In *Proceedings 17th ACM Symposium on Operating System Principles*, volume 34, pages 186–201, 1999.

[4] G. Allen, G. Aloisio, Z. Balaton, M. Cafaro, T. Dramlitsch, J. Gehring, T. Goodale, P. Kacsuk, A. Keller, H.-P. Kersken, T. Kielmann, H. Knipp, G. Lanfermann, B. Ludwiczak, L. Matyska, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, A. Reinefeld, M. Ruda, F. Schintke, E. Seidel, A. Streit, F. Szalai, K. Verstoep, and W. Ziegler. Early Experiences with the EGrid Testbed. In *IEEE/ACM International Symposium on Cluster Computing and the Grid CCGrid2001*, pages 130–127, May 2001.

[5] E. A. Brewer. When everything is searchable. *Communications of the ACM*, 44(3):53–54, Mar. 2001.

[6] Y. Chawathe, S. A. Fink, S. McCanne, and E. A. Brewer. A Proxy Architecture for Reliable Multicast in Heterogeneous Environments. In *Proceedings of the 6th ACM International Conference on Multimedia (Multimedia-98)*, pages 151–160, N.Y., Sept. 12–16 1998. ACM Press.

[7] Y. Chawathe, S. McCanne, and E. Brewer. An architecture for internet content distribution as an infrastructure service, 2000.

[8] S. Czerwinski, B. Y. Zhao, T. Hodes, A. D. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom-99*, pages 24–35, N.Y., Aug. 15–20 1999. ACM Press.

[9] DANTE. GEANT-MULTICAST. `http://www.dante.net/nep/GEANT-MULTICAST/`, 2002.

[10] K. Decker, K. Sycara, and M. Williamson. Middle-Agents for the Internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan, Aug. 1997.

[11] E. Guttman. Service Location Protocol: Automatic Discovery of IP Network Services. *IEEE Internet Computing*, 3(4):71–80, 1999.

[12] S. Helders. World Gazetteer. `http://www.world-gazetteer.com`, 2003.

[13] W. Hoschek. The Web Service Discovery Architecture. In *SC'2002 Conference CD*, Baltimore, MD, Nov. 2002. IEEE/ACM SIGARCH. pap161.

[14] A. Iamnitchi and I. Foster. On Fully Decentralized Resource Discovery in Grid Environments. *Lecture Notes in Computer Science*, 2242:51–??, 2001.

[15] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI-00)*, pages 197–212, Berkeley, CA, Oct. 23–25 2000. The USENIX Association.

[16] Z. Juhasz, A. Andics, and S. Pota. JM: A Jini Framework for Global Computing. In *Proceedings 2nd International Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, IEEE International Symposium on Cluster Computing and the Grid (CCGrid'2002)*, pages 395–400, Berlin, Germany, May 21 - 24 2002.

[17] S. Marti and V. Krishnan. Carmen: A Dynamic Service Discovery Architecture. Technical Report HPL-2002-257, Hewlett Packard Laboratories, Sept.23 2002.

[18] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, Hewlett Packard Laboratories, Mar.15 2002.

[19] B. Plale, P. Dinda, and G. von Laszewski. Key Concepts and Services of a Grid Information Service. In *Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems (PDCS 2002)*, `http://www.cs.northwestern.edu/ urgis/`, 2002.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Computer Communication Review*, volume 31, pages 161–172. Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley, 2001.

[21] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-Level Multicast Using Content-Addressable Networks. *Lecture Notes in Computer Science*, 2233:14–??, 2001.

[22] The Salutation Consortium. `http://www.salutation.org/`, 2002.

[23] Universal Plug and Play Forum. `http://www.upnp.org`, 1999.

[24] M. van Steen, F. Hauck, P. Homburg, and A. Tanenbaum. Locating objects in wide-area systems. *IEEE Communication*, 36(1):104–109, 1998.

[25] M. Wahl, T. Howes, and S. Kille. RFC 2251: Lightweight Directory Access Protocol (v3), Dec. 1997. Status: PROPOSED STANDARD.

[26] J. Waldo and K. Arnold. *The Jini specifications.* Jini technology series. Addison-Wesley, Reading, MA, USA, second edition, 2001. Rev. ed of: The Jini specification / Ken Arnold . . . [et al]. c1999.

Edited by: ¡names¿
Received: March 31, 2003
Accepted: June 5, 2003

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 35–44. http://www.scpe.org

ISSN 1895-1767
© 2005 SWPS

GRID-BASED VIRTUAL REALITY VISUALIZATION OF VOLUME DATA

PAUL HEINZLREITER , DIETER KRANZLMÜLLER , AND JENS VOLKERT*

Abstract. Grid computing evolves into a standard method for processing large datasets. Consequently most available grid applications focus on high performance computing and high-throughput computing. Using the Grid Visualization Kernel GVK, which is a grid middleware extension built on top of the Globus Toolkit, interactive visualization of the acquired simulation results can be performed directly on the grid. An example is the visualization of volume data within Virtual Reality environments, where the data for visualization is generated somewhere on the grid, while the user explores the visual representation at some other place on the grid.

Key words. grid computing, scientific visualization, interaction, virtual reality

1. Introduction. Grid computing is an approach enabling high-performance and high-throughput computing to exploit resources across organizational boundaries for calculation-intensive tasks in science and engineering [9]. Example application areas for grid computing are medical imaging, physics simulations, and satellite image processing. Due to the large amounts of data involved, such application's demand for computational power can barely be satisfied by using computational resources of only one organization.

For this reason grid computing offers persistent use of computational resources within virtual organizations spanning several countries or even continents. These virtual organizations are realized by grid middleware such as Globus [6], which deliver an abstraction of the underlying hardware, and therefore foster flexible and portable software development. Other middleware approaches such as Unicore [23], Cactus [3], or Legion [12] have also been proposed in the field of scientific data processing.

As an extension to these grid middleware toolkits, the Grid Visualization Kernel GVK aims at grid-enabled data visualization. Due to the nature of many grid applications, GVK's initial focus lies on the visualization of three-dimensional volume data as for example medical volume datasets. Providing such a flexible visualization component within a grid environment has major benefits such as flexibility concerning hardware due to the common grid middleware layer and direct access to visualization services running on the grid. For simulations exploiting the possibilities of grid computing, GVK offers a possibility to realize the required visualizations without being restricted to the usage of a specific visualization device. Therefore it supports various visualization devices including standard desktop workstations and even sophisticated Virtual Reality (VR) environments such as the CAVE [4]. The exploitation of virtual reality techniques is particularly useful in the field of scientific visualization due to the complexity of the involved datasets.

Comparable approaches to GVK are CAVERN [19], CUMULVS [11], WireGL [14], Chromium [15], VLAM-G [2], and Dv [1]. CAVERN mainly focuses on collaborative environments that consist of multiple interconnected VR devices. CUMULVS supports interactive distributed visualization of large data fields. WireGL is a distributed implementation of OpenGL, which enables parallel rendering. Chromium, which is originally based on WireGL is a high-performance rendering tool mainly targeted at homogeneous cluster platforms.

A globus-based distance visualization approach is presented in [8]. It is applied for remote rendering of tomographic images and relies heavily on Globus services.

VLAM-G [2] is a grid-based software environment focused at experimental science. It provides visualization capabilities to the user while hiding the complexity of the underlying grid infrastructure from the user.
Like GVK, Dv [1] also provides a visualization pipeline distributed over the grid. In contrast to GVK, Dv transmits the program to be executed within the visualization process together with the data from one grid node to the next.

Compared to these approaches, GVK is designed as a grid middleware component extending the possibilities of grid computing towards more efficient visualization methods that fully exploit the computational power of the underlying grid.

This paper describes the volume visualization capabilities and interaction modes of GVK. In Section 2 a brief overview of GVKs structure and VR visualization is given, while Section 3 describes the interaction in the context of scientific visualization offered by GVK. In Section 4 an overview over the data transmission mechanisms within GVK is provided whereas Section 5 describes necessary data transport optimizations. Section 6 describes an an

*GUP, Johannes Kepler University Linz, Altenbergerstr. 69, A-4040 Linz, Austria/Europe application example for grid-based interactive visualization. After that, Section 7 shows some results including rendered images and measurements before an outlook on future work concludes the paper.

The following section provides an overview of the general GVK structure and the VR visualization possibilities provided to the user.

2. The GVK Architecture for Virtual Reality Visualization. The design of the GVK architecture focuses on flexibility and efficiency, which are crucial to enable visualization on arbitrary devices connected to the grid. A major problem is posed by the limited network bandwidth available for transmitting visualization data. Support for optimized data transmission across grid nodes requires that GVK itself consists of various subsystems distributed over the nodes involved in the visualization. A block diagram containing the main modules of GVK and illustrating the data flow between them is shown in Figure 2.1. The GVK portal server

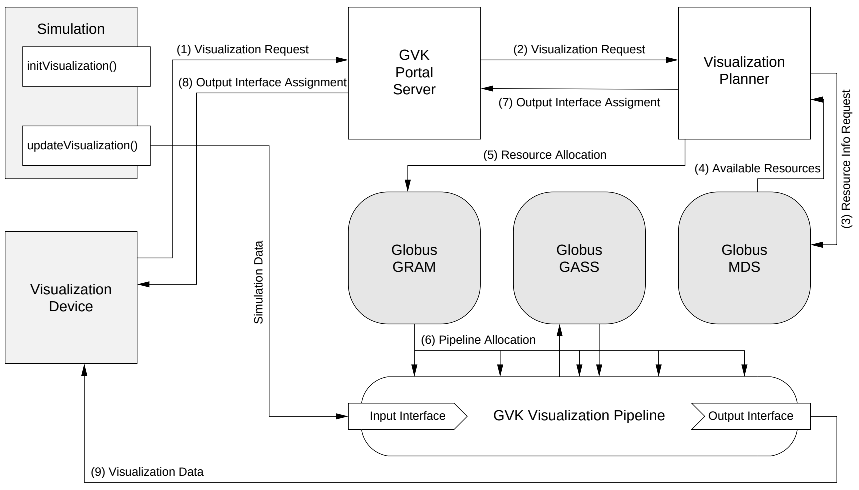


Fig. 2.1. *GVK operational structure*

receives the visualization request, which contains all required metadata like visualization type and input data location. The requests are forwarded to the visualization planner, which derives an appropriate visualization strategy based on the information provided by the visualization request and resource availability information obtained from Globus MDS [5]. In the next step the required resources are allocated and the visualization pipeline is constructed.

Later on, the visualization cycle is initialized by downloading the required data from the simulation server. The data is processed by sending it through the pipeline, which accomplishes the required data transformations (e.g. isosurface generation in case of volume visualization). Finally the generated graphics primitives are sent to the visualization device.

Supporting interaction with the user is achieved by collecting events from the visualization device, processing them in the main GVK subsystem and sending them back to the simulation server to reflect the interactive changes by the user within the simulation. Because network transportation optimizations are crucial to GVK, Globus IO is used as a relatively low level API which provides the required flexibility in network programming by providing a socket-type access.

An example visualization device connected to GVK is the CAVE [4], which enables the real-time visualization of interactive and complex data in a room-sized projection environment. The usage of such VR visualization techniques is expected to be useful for visualizing large amounts of data, because of the multiple dimensions and the immersion provided through stereo representation. OpenGL [24] was chosen for implementing the subsystem running on the visualization device because of its flexibility, its broad hardware support on desktop workstations, and easy integration of the CAVE and most other VR visualization devices.

At present, the GVK visualization system can cope with different types of input data:

- Three-dimensional volume data
- Two-dimensional grid data
- Arbitrary triangle meshes

Most of the input data processed by GVK is represented as three-dimensional volume data, due to the widespread use of this data format within the context of scientific visualization. To enable fast visualization of such data it is necessary to generate representations suitable to hardware-accelerated rendering such as triangulated isosurfaces or impostors. For the isosurface generation the well-know Marching Cubes algorithm [21] and the advanced Marching Tetrahedron algorithm are applied. Because of its computational requirements the isosurface triangulation is executed in the grid. The flexible representation of the volume data to be rendered by a voxel model allows the encoding of arbitrary additional information per voxel within the model. This is implemented by storing a string for each voxel which encodes required information, like for example density values, material properties, or flow field vectors. The necessary description of the string semantics is delivered to the GVK main subsystem by the visualization request. The visualization of two-dimensional grid data is done by triangulating the given data by means of adaptive mesh algorithms [13].

3. Interaction within GVK. An important feature of GVK is interactivity within the generated visualization scenarios. The devices which can be used for interaction depend on the type of visualization environment. Sitting in front of a conventional desktop workstation, the user is limited to keyboard and mouse in most cases. If a VR device like the CAVE is used, the interaction can be performed using devices that support six degrees of freedom like the wand, which is a three-dimensional mouse for virtual environments. Another possible extension is given by natural voice input [16]. There are two different modes of interaction:

- Local interaction
- Interaction with feedback to the simulation

Interaction techniques which can be handled locally on the visualization device are well-known techniques concerning changes of the observer's viewpoint and transformations of the whole model like scaling, rotation, and translation. These types of interaction can be processed autonomously on the visualization device without producing latency imposed by limited network bandwidth.

A more complex type of interaction affects the visual representation and the simulation data itself. At first the change requests are sent back to the GVK main subsystem, which processes them to calculate the necessary modifications for the simulation. Then the updated values are sent back to the simulation server. A drawback of this interaction mode is given by the requirement for the simulation to provide a specific interaction interface to GVK.

An example of such an interaction module is a generic voxel editor applicable to a wide range of three-dimensional volume data structures. The altered data structures are feed back to the remote simulation process. Afterwards, a new simulation cycle is invoked. Simultaneously the visualization is updated according to the users interaction. Thereby the user gets visual feedback about the effects of online data alteration. Consider for example a flooding simulation in which the user can virtually break a dam by successive removal of the voxels it is composed of. The editor supports following operations:

- Location-based selection
- Content-based selection
- Voxel deletion
- Voxel copy/cut/paste

Contrary to the straightforward implementation of the data modification operations, the selection has to be more sophisticated due to the huge amount of voxels composing a typical volume representation.

Content-based selection is a valuable tool if the user wants to visualize only a subset of the available data, which can be identified by its attributes. Given a input dataset consisting of a density value per voxel, one can select only a specific range of density values. Another interesting scenario from the medical application domain would be a set of volume data containing different types of body tissues. In this case content based selection enables exclusive rendering of the tissue type of interest.

Considering the location-based selection, there are several possibilities for selecting the desired area. Selection operations can be performed on single triangles as well as three-dimensional regions. For selecting single triangles the OpenGL picking mode can be applied for desktop and VR visualization devices. But in most cases the selection of a group of triangles is more desirable. The main problem within this approach is given by the lack of a 3D input device on desktop workstations. Taking into account the 2D mouse position and current projection matrix, one can derive a pick-ray which is intersected with the displayed volume model. With this approach the user can select a cylindrical area of interest by specifying diameter, startpoint, and length.

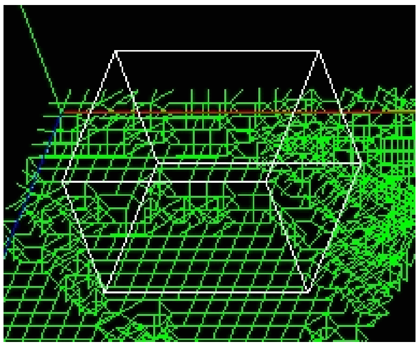


FIG. 3.1. *Selection Cube*

Another approach is given by a selection cube, as shown in Figure 3.1, where the user at first specifies the position of the cube center along the pick-ray, followed by the size of the cube. Within a VR environment, where a 3D mouse is available, the cube can be positioned directly by evaluating the location of the input device.

The selected triangles are sent to the main GVK subsystem which in turn determines the affected voxels. This can be accomplished by mapping the position of the selected triangles onto the voxel model. After applying the desired modifications to the selected voxels they are sent back to the GVK-enabled simulation, which then updates its internal data structures.

If the user requests a pasting operation of an e.g. previously copied region the semantics of that operation has to be specified, especially if the target region is not completely empty. The following actions are possible:

- Voxel replacement
- Voxel value addition
- Voxel value subtraction
- Filling of empty regions

A replacement of voxels in the target region is simple approach for handling a paste operation and for many use cases it is sufficient. For the value addition and substraction the content of the voxels is interpreted as a numerical value. The value after the paste operation is calculated out of the current voxel contents. If the target region is partially empty, only the empty voxels could be filled with the new values.

4. Data Transmission.

This section presents the different data transmission modes which are provided by GVK. To cope with the security issues arising within grid environments GVK relies on the security mechanisms provided by Globus such as GSS-API [7].

4.1. Basic GVK Communication.

The modules involved in a visualization are arranged to form an appropriate pipeline and get interconnected by GVK communication connections. The basic GVK communication functionality is provided by point-to-point connections, which are built on top of Globus IO. In addition to conventional data transmission the transmission over multiple connections based on threads as well as transmission of data described by a grammar which is sent in advance is provided. The sensibility of data transmission over multiple connections is illustrated in Figure 4.1. It shows a local minimum for the transmission time using eight connections which can be explained by the reduced probability of a random packet loss during transmission [18].

The transmission of typed data sends a structural description of the data as a preamble, so that the receiving side can at first build the appropriate data structures if they are not already agreed upon by the communication partners. Currently, the data format can contain basic integer and float data types as well as arrays and structures, which can be mutually nested in order to provide the appropriate description of the applications data.

4.2. Visualization Pipeline Construction .

The point-to-point connections used for communication are constructed according to the well-known client-server model. There are two necessary conditions for a visualization pipeline:

1. The server has to be available at client startup
2. Deadlocks must be avoided

Considering a more complex visualization pipeline with multiple connections for data transport and interaction, the startup order for the different modules is vital. To guarantee proper visualization pipeline construction the required modules are all started by a central instance, the visualization planner.

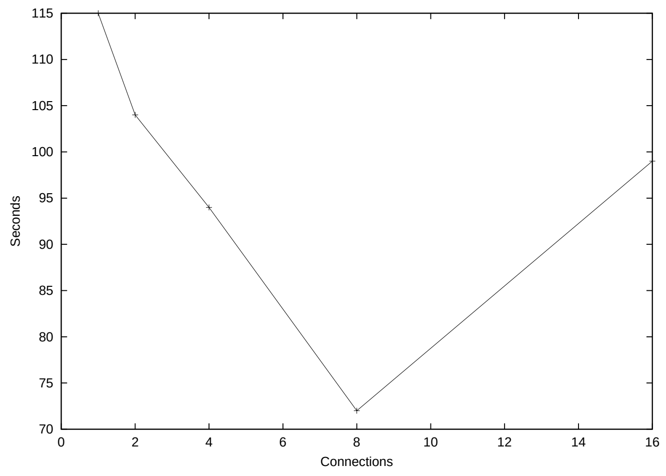


Fig. 4.1. *Transmission of 400 MB over multiple connections*

The distinction between client and server side of a connection is only required during pipeline construction in order to fulfill the first of the above conditions. During the visualization process each connection endpoint can send and receive data regardless of its client or server role.

Considering a simple visualization pipeline consisting of three modules the main challenge during visualization pipeline construction is given by the connections which send the interaction data describing the user input back to the simulation source.

For the GVK visualization planner, which constructs the pipeline, the visualization device is considered as a pure server, which means it is only handling server type connections. In contrast, the simulation data source is considered a pure client feeding input data into the visualization pipeline. The modules in between are hybrids acting as a server towards the the simulation data source and as a client on the visualization side. The main dataflow is directed from the client to the server side. With the interaction data being sent in the opposite direction its the other way around. The reason for the configuration is given by the requirement that for all connections between two components the client and server ends must be the same. This enables a working invocation sequence for the whole visualization pipeline starting at the visualization end.

In contrast to this strict requirements, the server and client roles which are presented to the user can be different. For example, if the user sends the visualization request from the visualization device, the device acts as client towards the GVK portal server.

4.3. Interaction Protocol. If an interactive visualization is required, the complexity of the visualization pipeline and the number of connections involved is increased significantly by adding dedicated connections for user input. In most cases these datastreams interconnect the same modules as the connections for visualization data.

An important issue for interactive visualizations is the reduction of the latency which is imposed by the network and the processing nodes involved. The following measures can reduce the impact of latency:

- Reduction of the number of involved modules
- Selective rendering of changed parts
- Reduction of visualization quality

Considering a visualization pipeline containing more than two modules and an interaction influencing the simulation data source at the other end of the visualization pipeline, the latency can be significantly reduced by immediately updating the visualization before forwarding the user input to the data source. This may lead to invalid representations until the data from the next simulation step is sent. The applicability of this approach is dependent on the type of the visualization pipeline and the timespan of a simulation step.

Selective rendering of updated parts provides an efficient technique for saving computation time and network bandwidth during the visualization process. This well-known approach is also applied in the area of graphical user interfaces in order to reduce the time spent during the redraw operation for a window being activated.

Within the context of a distributed rendering system like GVK it is even more beneficial. But during execution of a partial re-rendering operation it is important that the integration of the newly rendered area into the data already available at the visualization device can be done smoothly. An additional issue to be considered is the state of the simulation which is providing the data for the visualization. If the simulation has advanced too far during the user interaction, the whole representation has to be updated and sent through the visualization pipeline.

5. Optimizations within GVK. When transmitting large datasets for visualization purposes over the grid, network bandwidth inherently becomes the limiting factor to performance. Despite the substantial progress made in network bandwidth over the last years, todays Internet is far from being capable of transmitting large amounts of data as delivered by grid computing in reasonable time. For this reason several optimization techniques are provided by GVK to minimize interaction latency. Besides conventional data compression techniques [20], promising approaches are

- Occlusion culling
- Level-of-detail filtering
- Reference rendering and image warping

Occlusion culling [25] reduces the number of polygons to be transmitted by omitting the hidden parts of the model during the initial transmission steps. While the user is investigating the model, the rest of the polygons are transmitted. Level-of-detail algorithms [10] exploit hierarchical structures of triangle meshes by sending a coarse representation of the model first and finer representations afterwards. Reference rendering [22] uses pre-rendered images of defined viewing positions, which are sent to the visualization device on which to provide visual correct depictions for the given point of view at relatively high framerates. For more details on the applied optimization techniques, please refer to [17].

In order to implement all the mentioned optimizations it is necessary to use a low level programming interface to the data transmission functionality of Globus to provide maximum freedom to the implementation. Hence, the implementation is based on Globus IO.

6. Application Example. One application, which has been implemented to illustrate the feasibility of the GVK approach is a distributed volume rendering application consisting of three components:

- A data source providing volume data
- A data processing module doing isosurface extraction
- An interactive triangle mesh rendering component

The input data which enters the visualization pipeline consists of volume data represented by a set of voxels. Different types of input data have been tested, like a digital terrain model and several different three-dimensional geometric primitives.

The data processing module performs isosurface extraction based on the marching cubes or marching tetrahedron algorithm on the volume data and extracts the isosurface requested by the user. The obtained triangle meshes are then stored on the data processing server to speed up future access from possibly multiple clients.

To reduce the load on the network as well as the required computation time a level of detail approach has been embedded into the isosurface extraction algorithm. The resolution of the produced triangle mesh is dependent on the size of the marching cube used within the isosurface extraction. After extraction each level of detail is stored on the dataserver and sent to the registered visualization clients upon request.

The visualization client receives the triangle mesh from the data processing module and presents it to the user by rendering on a desktop workstation or in the CAVE. Available rendering modes include wireframe representation, flat, and smooth shading.

The volume rendering pipeline supports sophisticated interaction possibilities like cut, copy and paste operations with different semantics like adding, substracting, and replacing the data in the paste region.

The selection is performed by a cube-type selector as shown in Figure 3.1. Additional user interface functionality includes Undo and Redo. This operations are acomplished by storing the intermediate visualization results in Undo and Redo buffers within the data processing module and restoring them upon request.

7. Results. This section shows a few rendered images of test datasets including geometric primitives and a heightfield. Figure 7.1 represents a collection of geometric objects triangulated with the marching tetrahedron algorithm and rendered using smooth shading, whereas Figure 7.2 shows a visualization done in wireframe mode. Figure 7.3 gives an example for a heightfield visualization representing an terrain elevation map.

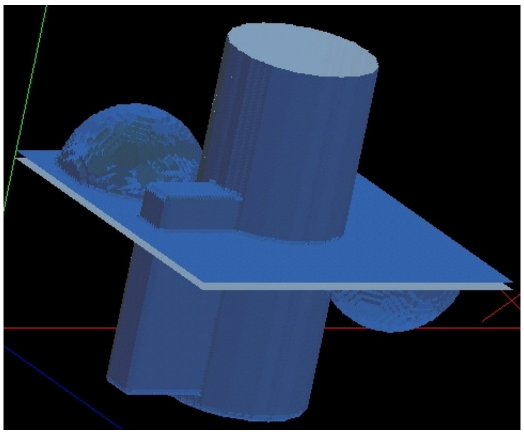


Fig. 7.1. *Geometric objects with smooth shading*

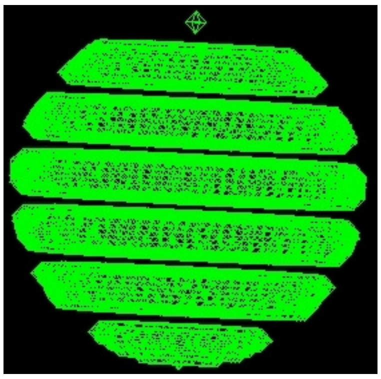


Fig. 7.2. *Wireframe representation of a sliced sphere*

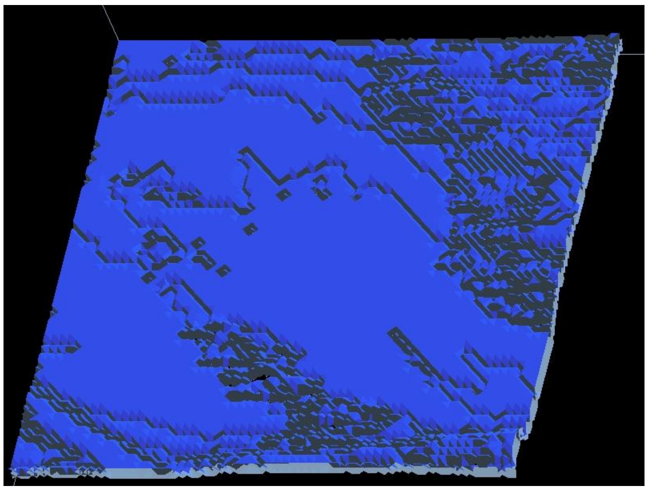


Fig. 7.3. *Elevation map*

Figures 7.4 and 7.5 illustrate runtime measurements for the triangulation part. The input data model used for the measurements is shown in Figure 7.1. The results are represented by two graphs, one for the marching cube and the other for the marching tetrahedron algorithm. A comparison of the two graphs in Figure 7.4 illustrates that the marching tetrahedron algorithm produces significantly more triangles than the marching cube algorithm. This leads to better triangulation without ambiguities but also results in a higher load on the network when the triangles are sent to the visualization device. These measurements also illustrate the importance of triangle count reduction for visualization speedup.

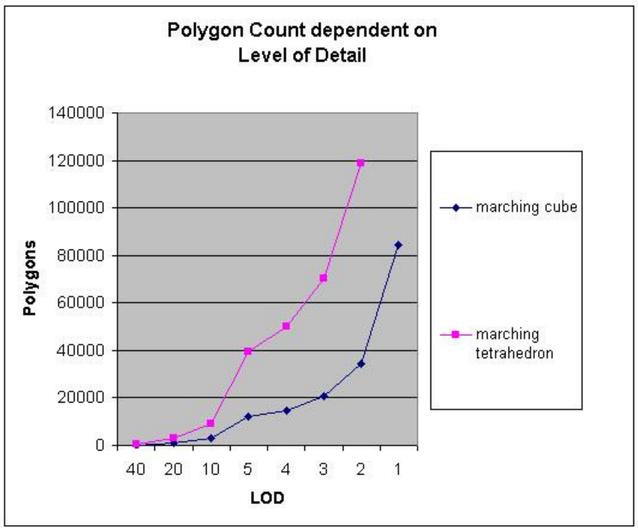


Fig. 7.4. *Dependency between polygon count and level of detail*

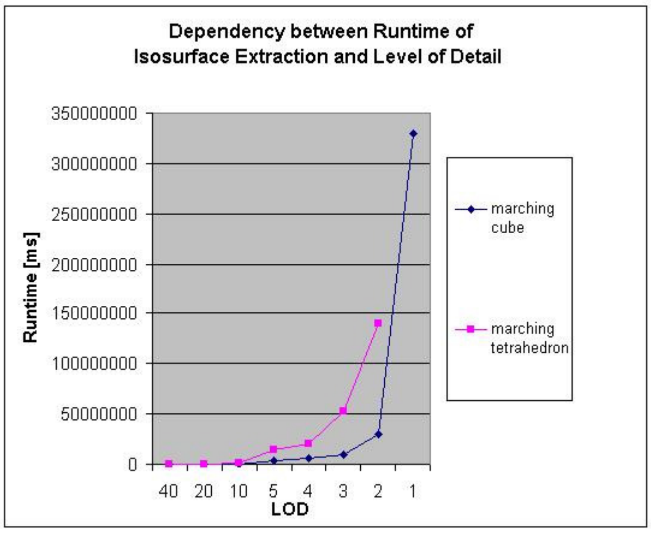


Fig. 7.5. *Dependency between runtime and level of detail*

Figure 7.6 illustrates the latency within an interactive rendering pipeline dependent on the number of triangles which are processed. The pipeline consists of three nodes, a data source sending out an array of heightfield data, a triangulation module which produces the triangulation of the heightfield, and a rendering module which is displaying the triangle mesh in the CAVE. The diagram shows the time between the user issuing a request and the visualization being updated. In this case the user request is sent back over the triangulation node to the data source which then feeds the changed data into the pipeline.

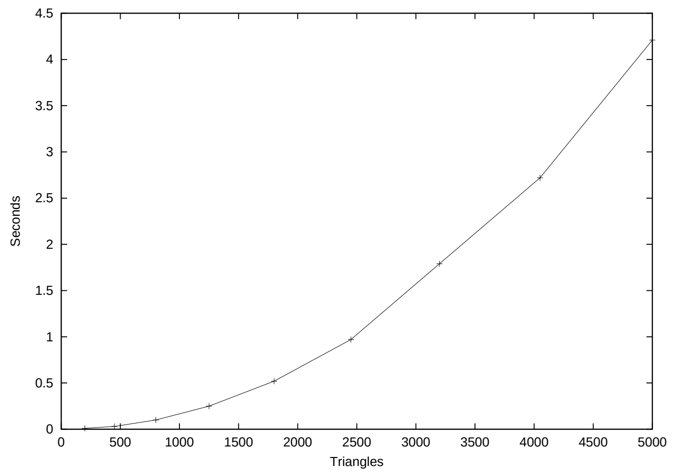


FIG. 7.6. *Interaction latency*

8. Conclusions and Future Work. The Grid Visualization Kernel is a grid middleware component for visualization purposes that supports various types of input data and visualization devices. In contrast to existing distributed visualization solutions, GVK uses the computational power available on the grid for visualization optimizations. GVK is currently available as a first prototype with only limited data processing and visualization features, like VR represention in the CAVE and network transport optimization by using standard data compression algorithms and level of detail reduction of triangle meshes. Available visualization devices include triangle mesh renderers and an image based visualization tool.

Future work will focus on extending the flexibility of GVK as well as dynamic adaptation to varying bandwidths. Another important issue will be a more flexible interaction model which can be applied to different applications.

Acknowledgments. The work on GVK presented in this paper was supported by our colleagues, most notably Hans-Peter Baumgartner, Peter Praxmarer, Herbert Rosmanith, Andreas Wasserbauer, and Gerhard Kurka.

The Grid Visualization Kernel (GVK) is partially supported by the Crossgrid Project of the IST Program of the European Commission under contract number IST-2001-32243.

REFERENCES

[1] M. Aeschlimann, P. Dinda, J. López, B. Lowekamp, L. Kallivokas, D. O'Halleron, *Preliminary Report on the Design of a Framework for Distributed Visualization*, Proc. PDPTA '99, Las Vegas, Nevada, USA, pp. 1833–1839, 1999

[2] H. Afsarmanesh, R. Belleman, A. Belloum, A. Benabdelkader, J.F.J. van den Brand, T.M. Breit, H. Bussemaker, G. Eijkel, A. Frenkel, C. Garita, D.L. Groep, A.W. van Halderen, R.M.A. Heren, Z.W. Hendrikse, L.O. Hertzberger, J. Kaandorp, E.C. Kaletas, V. Klos, P. Sloot, R.D. Vis, A. Visser, H.H. Yakali, *VLAM-G: A Grid-Based Virtual Laboratory*, Scientific Programming Journal, Vol. 10, No. 2, pp. 173–181, 2002

[3] G. Allen, T. Goodale, E. Seidel, *The cactus computational collaboratory: Enabling technologies for relativistic astrophysics,and a toolkit for solving pdes by communities in science and engineering*, 7th Symposium on the Frontiers of Massively Parallel Computation-Frontiers 99, IEEE, New York 1999

[4] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, J. C. Hart, *The CAVE: audio visual experience automatic virtual environment*, Communications of the ACM, Vol. 35, No. 6, pp. 64–72, 1992

[5] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, *Grid Information Services for Distributed Resource Sharing*, Proc. of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001

[6] I. Foster, C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, Intl. Journal of Supercomputer Applications, Vol. 11, No. 2, pp. 4–18, 1997

[7] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, *A Security Architecture for Computational Grids*, Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83–92, 1998

[8] I. Foster, J. Insley, G. von Laszewski, C. Kesselman, M. Thiebaux, *Distance Visualization: Data Exploration on the Grid*, IEEE Computer Magazine, Vol. 32, No. 12, pp. 36–43, 1999

[9] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid - Enabling Scaleable Virtual Organizations*, Intl. Journal of Supercomputer Applications, Vol. 5, No. 3, 2001

[10] M. Garland, P. S. Heckbert, *Surface Simplification Using Quadric Error Metrics*, Computer Graphics, Vol. 31, pp. 209–216, 1997

[11] G. A. Geist, J. A. Kohl, P. M. Papadopoulos, *CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications*, Intl. Journal of High Performance Computing Applications, Vol. 11, No. 3, pp. 224–236, August 1997

[12] A. Grimshaw, A. Ferrari, F. Knabe, M. Humphrey, *Legion: An Operating System for Wide-Area Computing*, IEEE Computer, Vol. 32, No. 5, pp. 29–37, 1999

[13] H. Hoppe, *Progressive meshes*, Proc. ACM SIGGRAPH '96, pp. 99–108, 1996

[14] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, P. Hanrahan, *WireGL: a scalable graphics system for clusters*, Proc. ACM SIGGRAPH '01, pp. 129–140, 2001

[15] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P.D. Kirchner, J.T. Klosowski, *Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters*, Proc. ACM SIGGRAPH '02, pp. 693–207, San Antonio, Texas, USA, August 2002

[16] D. Kranzlmüller, B. Reitinger, I. Hackl, J. Volkert, *Voice Controlled Virtual Reality and Its Perspectives for Everyday Life*, Proc. APC 2001, Arbeitsplatzcomputer 2001, Fachtagung der GI/ITG Fachgruppe APS+PC zum Thema "Pervasive Ubiquitous Computing", VDE-Verlag, Technical University Munich, Munich, Germany, pp. 101–107, 2001

[17] D. Kranzlmüller, G. Kurka, P. Heinzlreiter, J. Volkert, *Optimizations in the Grid Visualization Kernel*, Proc. of PDIVM 2002, Workshop on Parallel and Distributed Computing in Image Processing, Video Processing and Multimedia, IPDPS 2002 Symposion, Ft. Lauderdale, Florida, April 2002

[18] J. Lee, D. Gunther, B. Thierney, B. Allcock, J. Bester, J. Bresnahan, S. Tuecke, *Applied Techniques for High Bandwidth Data Transfers Across Wide Area Networks*, Proc. of International Conference on Computing in High Energy and Nuclear Physics, Bejing, China, September 2001

[19] J. Leigh, A. E. Johnson, T. A. DeFanti, *CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments*, Journal of Virtual Reality Research, Development and Applications, Vol. 2.2, pp. 217–237, the Virtual Reality Society 1997

[20] D. A. Lelewer, D. S. Hirschberg, *Data Compression*, ACM Computing Surveys (CSUR), Vol. 19, No. 3, pp. 261–296, 1987

[21] W. E. Lorensen, H. E. Cline, *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*, Proc. ACM SIGGRAPH '87, pp. 163–169, 1987

[22] L. McMillan, G. Bishop, *Plenoptic modeling: an image-based rendering system*, Proc. ACM SIGGRAPH '95, pp. 39–46, 1995

[23] M. Romberg, *The Unicore Architecture: Seamless Access to Distributed Resources*, Proc. 8th IEEE Symposion on High Performance Distributed Computing, HPDC 1999, pp. 287–293, Los Alamitos, California, August 1999

[24] M. Woo, J. Neider, T. Davis, D. Shreiner, *OpenGL Programming Guide*, Third Edition, Addison Wesley, 1999

[25] H. Zhang, D. Manocha, T. Hudson, K. Hoff, *Visibility Culling using Hierarchical Occlusion Maps*, Proc. ACM SIGGRAPH'97, pp. 77–88, 1997

Edited by: Zsolt Nemeth, Dieter Kranzlmuller, Peter Kacsuk
Received: April 7, 2003
Accepted: June 5, 2003

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 45–57. http://www.scpe.org

ISSN 1895-1767
© 2005 SWPS

ON THE EXTENSION AND APPLICABILITY OF THE P-GRAPH MODELING PARADIGM TO SYSTEM-LEVEL DIAGNOSTIC PROBLEMS

BALÁZS POLGÁR[†] , ENDRE SELÉNYI[†] , AND TAMÁS BARTHA[‡]

Abstract.
This paper presents a novel approach that formulates different types of diagnostic problems similarly. The main idea is the reformulation of the diagnostic procedures as P-graph models. In this way the same paradigm can be applied to model different aspects of a complex problem. The idea is illustrated by solving the probabilistic diagnosis problem in multiprocessor systems and by extending it with some additional properties. Thus, potential link errors and intermittent faults are taken into consideration and the comparator based diagnostics is formulated including potential comparator errors.

Key words. multiprocessor systems, fault diagnosis, maximum likelihood diagnostics, modeling with P-graphs

1. Introduction. Diagnostics is one of the major tools for assuring the reliability of complex systems in information technology. In such systems the test process is often implemented on system level: the "intelligent" components of the system test their local environment and each other. The test results are collected, and based on this information the good or faulty state of each component is determined. This classification procedure is known as *diagnostic process*.

The early approaches that solve the diagnostic problem employed oversimplified binary fault models, could only describe homogeneous systems, and assumed the faults to be permanent. Since these conditions proved to be impractical, lately much effort has been put into extending the limitations of traditional models [1]. However, the presented solutions mostly concentrated on a single aspect of the problem.

In this paper we present a novel modeling approach based on P-graphs that can integrate these extensions in one framework, while maintaining a good diagnostic performance. With this model, we formulate diagnosis as an optimization problem and apply the idea to the well-known multiprocessor testing problem. Furthermore, we have not only integrated existing solution methods, but proceeding from a more general base we have extended the set of solvable problems with new ones.

The paper is structured as follows. First an overview is given about the traditional aspects of system-level diagnosis [2, 3, 4] and the generalized test invalidation model used in our approach. Afterwards, the diagnostic problem of a multiprocessor system is formulated with the use of P-graphs [5]. In the fourth section two supplements are presented which can accelerate the solution method. Both use additional a priori information. The first one adds unit failure probabilities to the model, the second utilizes special knowledge about the structure of the system. Then an important aspect, the extensibility of the model is demonstrated via some examples. The generation and the solution method of a P-graph model and the acceleration techniques are clarified on a small example and simulation results are presented. Finally, we conclude and sketch the direction of future work.

2. System-Level Diagnosis. System-level diagnosis considers the *replaceable units* of a system, and does not deal with the exact location of faults within these units. A *system* consists of an interconnected network of independent but cooperating *units* (typically processors). The fault state of each unit is either *good* when it behaves as specified, or *faulty*, otherwise. The *fault pattern* is the collection of the fault states of all units in the system. A unit may test the *neighboring* units connected with it via direct links. The network of the units testing each other determines the *test topology*. The outcome of a test can be either *passed* or *failed* (denoted by 0/1 or G/F); this result is considered *valid* if it corresponds to the actual physical state of the tested unit.

The collection of the results of every completed test is called the *syndrome*. The test topology and the syndrome are represented graphically by the *test graph*. The vertices of a test graph denote the units of the system, while the directed arcs represent the tests originated at the *tester* and directed towards the *tested unit* (UUT). The result of a test is shown as the label of the corresponding arc. Label 0 represents the passed test result, while label 1 represents the failed one. See Figure 2.1 for an example test graph with three units.

[†]Dept. of Measurement and Inf. Systems, Budapest Univ. of Technology and Economics, Magyar Tudósok krt. 2, Budapest, Hungary, H-1117, (polgar@mit.bme.hu, selenyi@mit.bme.hu).
[‡]Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende u. 13–17, Budapest, Hungary, H-1111, (tamas.bartha@sztaki.hu).

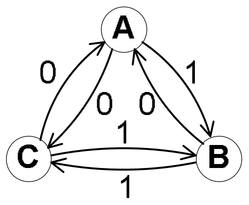


Fig. 2.1. *Example test graph (test topology with syndrome)*

2.1. Traditional Approaches. Traditional diagnostic algorithms assume that

(i) faults are permanent,
(ii) states of units are binary (*good*, *faulty*),
(iii) the test results of good units are always valid,
(iv) the test results of faulty units can also be invalid. The behavior of faulty tester units is expressed in the form of *test invalidation models*.

Fig. 2.2 shows the fault model of a single test and Table 2.1 covers the possible test invalidation models, where the selection of c and d values determines a specific model. The most widely used example is the so-called PMC (Preparata, Metze, Chien) test invalidation model, ($c = any, d = any$) which considers the test result of a faulty tester to be independent of the state of the tested unit. According to another well-known test invalidation model, the BGM (Barsi, Grandoni, Maestrini) model ($c = any, d = faulty$) a faulty tester will always detect the failure of the tested unit, because it is assumed that the probability of two units failing the same way is negligible.

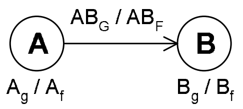


Fig. 2.2. *Fault model of a single test*

Table 2.1
Traditional test invalidation models

State of tester	**State of UUT**	**Test result**
good	*good*	*passed*
good	*faulty*	*failed*
faulty	*good*	$c \in \{passed, failed, any\}$
faulty	*faulty*	$d \in \{passed, failed, any\}$

The purpose of system-level diagnostic algorithms is to determine the fault state of each unit from the syndrome. The difficulty comes from the possibility that a fault in the tester processor invalidates the test result. As a consequence, multiple "candidate" diagnoses can be compatible with the syndrome. To provide a complete diagnosis and to select from the candidate diagnoses, the so-called *deterministic* algorithms use extra information in addition to the syndrome, such as assumptions on the size of the fault pattern or on the testing topology.

Alternatively, *probabilistic* algorithms try to determine the most probable diagnosis assuming that a unit is more likely good than faulty [6]. Frequently, this maximum likelihood strategy can be expressed simply as "many faults occur less frequently than a few faults." Thus, the aim of diagnostics is to determine the minimal set of faulty elements of the system that is consistent with the syndrome.

2.2. The Generalized Approach. In our previous work [5, 7] we used a generalized test invalidation model, introduced by Blount [8]. In this model, probabilities are assigned to both possible test outcome for each combination of the states of tester and tested unit (Table 2.2). Since the good and faulty results are complementary events, the sum of the probabilities in each row is 1. The assumption of the complete fault coverage can be relaxed in the generalized model by setting probability p_{b1} to the fault coverage of the test.

Probabilities p_{c0}, p_{c1}, p_{d0} and p_{d1} express the distortion of the test results by a faulty tester. Moreover, the generalized model is able to encompass *false alarms* (a good tester finds a good unit to be faulty) by setting probability p_{a1} to nonzero, however, it is not a typical situation.

Table 2.2
Generalized test model

State of tester	**State of UUT**	**Probability of test result**	
		0	**1**
good	*good*	p_{a0}	p_{a1}
good	*faulty*	p_{b0}	p_{b1}
faulty	*good*	p_{c0}	p_{c1}
faulty	*faulty*	p_{d0}	p_{d1}

Of course, the generalized test invalidation model covers the traditional models. Setting the probabilities as $p_{a0} = p_{b1} = 1$, $p_{c0} = p_{c1} = p_{d0} = p_{d1} = 0.5$, and $p_{a1} = p_{b0} = 0$, the generalized model will have the characteristics of the PMC model, while the configuration $p_{a0} = p_{b1} = p_{d1} = 1$, $p_{c0} = p_{c1} = 0.5$ and $p_{a1} = p_{b0} = p_{d0} = 0$ will make it behave like the BGM model. Analogically, every traditional test invalidation model can be mapped as a special case to our model.

3. Diagnosis Based on P-Graphs. The name 'P-graph' originates from the name 'Process-graph' from the field of Process Network Synthesis problems (PNS problem for short) in chemical engineering. In connection with this field the mathematical background of the solution methods of PNS problems have been well elaborated, see [9, 10, 11].

3.1. Definition of the P-Graph Model of the Diagnostic System. A P-graph is a directed bipartite graph. Its vertices are partitioned into two sets, with no two vertices of the same set being adjacent. In our interpretation one of the sets contains *hypotheses* (assumptions or information about the state of units and the possible test results), the otherone contains *logical relations* between the hypotheses. Hypotheses are represented by solid dots and logical relations by short horizontal lines. The edges of the graph point from the *premisses*[1] 'through' the logical relation to the *consequences*[2].

The set of premisses contains all states of each unit (e.g., 'unit A is good', 'unit A is faulty', 'unit B is good', denoted by A_g, A_f, B_g), and the set of consequences contains the test results (e.g. 'unit A finds unit B to be good', 'unit B finds unit C to be faulty', denoted by AB_G, BC_F). Logical relations determine the possible premisses of each possible test result. This means there are 8 logical relations for each test according to the 8 possible combinations of the state of tester, the state of the tested unit and the possible test results. Probabilities in Table 2.2 are assigned to relations expressing the uncertainty of the consequences. The P-graph model of a single-test fault model introduced on Fig. 2.2 can be seen on Fig. 3.1.

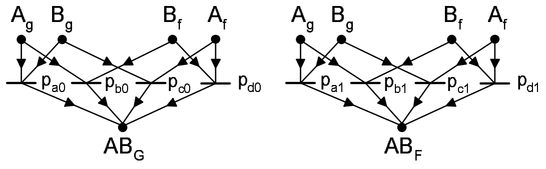


Fig. 3.1. *P-graph model of a single test (vertices with same label represent a single vertex; multiple instances are only for better arrangement)*

A *solution structure* is defined as a subgraph of the original P-graph, which deduces the consequences back to a subset of premisses.

Constraints can be defined in the model in order to assure that in a solution structure a unit should have one and only one state. Formally, for each hypothesis h the function $\epsilon(h)$ determines the set of hypotheses which are excluded by h. A P-graph is *consistent* if all constraints are satisfied.

The *probability of the syndrome (P_S)* is the product of probabilities of relations in a solution structure. This is the probability of occurrence of the consequences under the conditions of the given subset of system premisses, that is the probability of occurrence of the syndrome under the condition of a given fault pattern.

[1]premiss: preliminary condition
[2]i.e., there are edges from the premisses to the logical relation and from the logical relation to the consequences.

During the solution process more consistent solution structures can exist having different subsets of premisses and having different P_S values. The object is to find the solution structure containing the subset of premisses that implies the known consequences with maximum likelihood. This is an optimization task.

In principle, this task can be solved by general mathematical programming methods like mixed integer non-linear programming (MINLP), however, they are unnecessary complex. Friedler et al. [9, 10, 11] developed a new framework for solving PNS problems effectively by exploiting the special structure of the problem and the corresponding mathematical model.

3.2. Steps of the Solution Algorithm.

1. The maximal P-graph structure is generated. It contains only the relevant hypotheses and the relevant logical relations, but constraints are not yet satisfied. It contains all possible fault patterns being consistent with the given syndrome.
2. Every combinatorially feasible solution structure is obtained. These are the structures that satisfy the constraints and draw the consequences (i.e., the syndrome) back to a subset of the premisses. Each of these subsets determines a possible fault pattern.
3. For each combinatorially feasible solution structure the probability of syndrome is calculated. This is the conditional probability of the syndrome under the condition of a particular fault pattern.
4. The structure having the highest probability is selected; this solution structure contains the maximum-likelihood diagnosis.

Steps 2–4 can be completed either by a general solver for linear programming (since the generated maximal structure is a special flat P-graph), or with an adapted SSG algorithm [9]. Since the complexity of the generation of every combinatorially feasible solution structure in step 2 is exponential, the use of some kind of branch and bound technique can be employed to accelerate the solution method.

In a branch and bound algorithm a search tree is built. It branches as possible exclusive premisses are fixed for consequences. After a consistent solution is found, all those branches are bounded, which cannot have better P_S value. The algorithm proceeds until all branches are either containing a solution structure or are bounded. The algorithm can be more efficient if the first solution structure have been found quickly and it is near to the optimal one, because in this case bigger branches can be bounded.

4. Supplements to the Model.

The base model described in the previous section can be supplemented or altered in order to increase the efficiency of the solution algorithm. In this section two enhancement methods are presented.

4.1. Probabilities of Unit Failures.

Embedding more *a priori information* into the model does not necessarily increase the diagnostic accuracy, but it can speed up the solution algorithm. This is the case if unit failure probabilities are taken into account.

Let's define p_{Ag} and $p_{Af} = 1 - p_{Ag}$ as the *probability of the good* and the *faulty* state of unit A. Similarly, probabilities for the states of other units are defined. If the system is homogeneous, the values are the same for each unit.

In the model these values are assigned to the vertices representing the corresponding state information of the unit (Fig. 4.1). It is similar, as probabilities were assigned to logical relations. Now the *probability of the syndrome (P_S)* is defined as the *product of probabilities of relations and premisses* being in the solution structure.

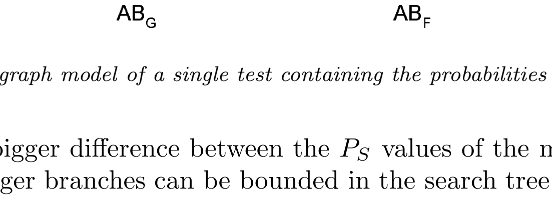


FIG. 4.1. *P-graph model of a single test containing the probabilities of the states*

This addition results in a bigger difference between the P_S values of the more and the less probable fault patterns, which results that bigger branches can be bounded in the search tree.

4.2. Mutual Tests. For special structures the model can be simplified or altered in order to increase the efficiency of the solution algorithm utilizing the extra information known about the system. An example for it is the case of mutual tests.

Tests t_1 and t_2 are *mutual*, if the tester in t_1 is the tested unit in t_2 and the tested unit in t_1 is the tester in t_2. Because the sets of possible premisses of the results of these tests are the same, the two tests can be handled together. Let's substitute the two tests with one mutual test having four possible test results (*GG,GF,FG,FF*)(Fig. 4.2). The test invalidation model is modified according to Table 4.1.

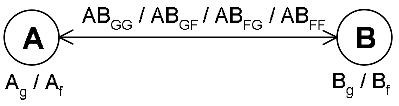


Fig. 4.2. *Fault model of a mutual test*

Table 4.1
Probabilities of test result pairs depending on the state of units

State of A	State of B	Probability of test result pair			
		AB_{GG}	AB_{GF}	AB_{FG}	AB_{FF}
good	*good*	p_{A0}	p_{A1}	p_{A2}	p_{A3}
good	*faulty*	p_{B0}	p_{B1}	p_{B2}	p_{B3}
faulty	*good*	p_{C0}	p_{C1}	p_{C2}	p_{C3}
faulty	*faulty*	p_{D0}	p_{D1}	p_{D2}	p_{D3}

In the initial P-graph model the 2x2x4 logical relations (two pieces from the P-graph on Fig. 3.1) are replaced with 4x4 relations (Fig. 4.3). Although the size of the initial model is unchanged, after step 1 in solution algorithm the maximal structure contains only 4 relations instead of 2x4, because information is in a compact form in this model.

Of course, probabilities in Table 4.1 can be derived from the previous ones being in Table 2.2:

$$\begin{array}{llll}
p_{A0} = p_{a0}^2 & p_{A1} = p_{a0}p_{a1} & p_{A2} = p_{a1}p_{a0} & p_{A3} = p_{a1}^2 \\
p_{B0} = p_{b0}p_{c0} & p_{B1} = p_{b0}p_{c1} & p_{B2} = p_{b1}p_{c0} & p_{B3} = p_{b1}p_{c1} \\
p_{C0} = p_{c0}p_{b0} & p_{C1} = p_{c0}p_{b1} & p_{C2} = p_{c1}p_{b0} & p_{C3} = p_{c1}p_{b1} \\
p_{D0} = p_{d0}^2 & p_{D1} = p_{d0}p_{d1} & p_{D2} = p_{d1}p_{d0} & p_{D3} = p_{d1}^2
\end{array}$$

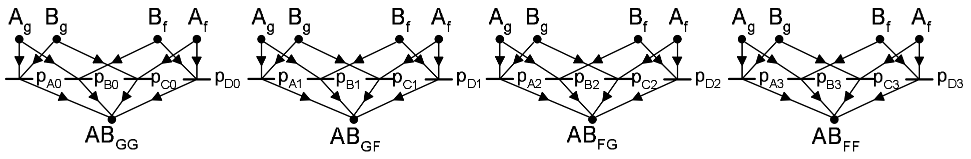


Fig. 4.3. *P-graph model of a mutual test*

5. Extensions of the Model. The main contribution of this novel modeling approach is its generality. With its use several aspects of system-level diagnosis can be handled in the same framework. Furthermore, it became possible to formulate new aspects of diagnosis. Thus, it is possible to model and diagnose for instance the following cases:

Systems with heterogeneous elements. There are systems like the *supercomputer APEMille* [12] which are built up from processing elements having different complexity and different behavior. These differences appear as the differences between the test invalidation models of the components. In the model it can be handled easily. Each element can have its own test invalidation model and the probability for the result of a test is taken from the invalidation model of the tester.

Multiple fault states. It is possible to construct and handle a finer model of the state of a unit than the binary one. This means that the failure modes of a system can be distinguished as in the fault model of the *Parsytec GCel massively parallel multiprocessor machine* described in [13]. In this model processors can have three states, namely *good* if it operates as expected, *faulty* if it operates but provides faulty results and *dead* if it doesn't operate or doesn't communicate. This also implies that the result of a test can have more than two values as well.

The model of a system having mutual tests is an example for systems having multiple test results. To take multiple fault states into account, rows should be added to the test invalidation table according to the possible combinations of the states of the tester and tested unit. This will result in more logical relations in the P-graph.

Intermittent faults. These are permanent faults that become activated only in special circumstances. Because these circumstances are usually independent from the testing process, these type of faults are diagnosed for instance on the basis of multiple syndromes as in the *method of Lee and Shin* [14].

Systems with potential link errors. The base model assumes that links between processors are working always properly. Conversely, the probability of the error of a link is not negligible in such systems where processors are connected to each other through routers as in the above mentioned *Parsytec GCel machine.*

Systems based on the comparator model. The comparator based diagnostic model of multiprocessor systems [15] is an alternative to the tester–tested unit model introduced by Preparata et al. In this model both units perform the same test and a comparator compares the bit-sequence of the outputs. In this case the syndrome consists of the results of the comparators, namely the information that *'the two units differ'* or *'the two units operate similarly'.* Of course, this model can be applied only for homogeneous systems.

An example for the comparator based model is the previously mentioned commercially available *APEMille supercomputer* [12] which was developed in collaboration by IEI-CNR of Rome and Pisa, and the DESY Zeuthen in Germany.

A further possible application field of this model is the *wafer scale diagnosis* [15, 16]. The idea is to connect individual processors on the wafer in order to form a multiprocessor system just for the time of the diagnostic phase of the production. The advantage of it is that in this case processors can be tested on working speed—and not only on reduced speed—before packaging and the more faulty processors identified before packaging results in less cost.

The models of the last three items in the list are presented in details in the next subsections.

5.1. Modeling Intermittent Faults. Although handling of intermittent faults is one of the difficult to manage diagnostic problems, a possible solution is the use of multiple syndromes, as mentioned above. In this approach two or more testing rounds are performed in a row, and the possible differences between the subsequent syndromes are used to detect intermittent faults.

The adaptation of the diagnostic P-graph model to this approach is similar to the model of mutual tests. Considering the case of double syndromes the fault model, the test invalidation model and the P-graph model correspond to the appropriate models of mutual tests on Fig. 4.2, 4.3 and in Table 4.1 having differences in the testing method (Fig. 5.1) and in the derivation method of probability parameters.

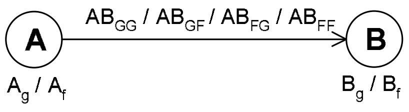


FIG. 5.1. *Fault model of a single test in case of double syndromes*

$$\begin{array}{llll}
p_{A0} = p_{a0}^2 & p_{A1} = p_{a0}p_{a1} & p_{A2} = p_{a1}p_{a0} & p_{A3} = p_{a1}^2 \\
p_{B0} = p_{b0}^2 & p_{B1} = p_{b0}p_{b1} & p_{B2} = p_{b1}p_{b0} & p_{B3} = p_{b1}^2 \\
p_{C0} = p_{c0}^2 & p_{C1} = p_{c0}p_{c1} & p_{C2} = p_{c1}p_{c0} & p_{C3} = p_{c1}^2 \\
p_{D0} = p_{d0}^2 & p_{D1} = p_{d0}p_{d1} & p_{D2} = p_{d1}p_{d0} & p_{D3} = p_{d1}^2
\end{array}$$

In the general case n test rounds are performed in a row. Thus, 2^n result combinations are in the fault model and each contains n single test result. Consider the case when a result combination contains n_G *passed* and $n_F = n - n_G$ *failed* test results. In this case the corresponding column in the test invalidation model contains the derived probabilities $p_{a0}^{n_G} \cdot p_{a1}^{n_F}$, $p_{b0}^{n_G} \cdot p_{b1}^{n_F}$, $p_{c0}^{n_G} \cdot p_{c1}^{n_F}$, $p_{d0}^{n_G} \cdot p_{d1}^{n_F}$.

Although the model generated this way seems to be growing as n increases, the P-graph model to be solved (the model created after 1st solution step) is of the same size. The reason is that although the information known about the system is increased, it is represented in a compact form in the probabilities while the size of the syndrome remains unchanged.

5.2. Modeling Systems with Potential Link Errors. The fault model of a single test shown on Fig. 2.2 is extended with the *Link* component (L) according to Fig. 5.2. For each AB test a separate link L_{AB} is assumed, which has either good or faulty state (denoted by L_{ABg}, L_{ABf}).

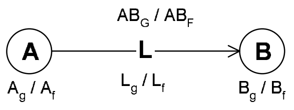


Fig. 5.2. *Fault model of a single test with potential link error*

The state of the link influences the test result, therefore the test invalidation table is modified according to Table 5.1. The probabilities are the same as in Table 2.2 if L_{AB} is good, but additional parameters are introduced if it is faulty.

Table 5.1
Probabilities of test results when considering the state of the link

State of L	**State of A**	**State of B**	**Probability of test result**	
			AB_G	AB_F
good	*good*	*good*	p_{a0}	p_{a1}
good	*good*	*faulty*	p_{b0}	p_{b1}
good	*faulty*	*good*	p_{c0}	p_{c1}
good	*faulty*	*faulty*	p_{d0}	p_{d1}
faulty	*good*	x	p_{e0}	p_{e1}
faulty	*faulty*	x	p_{f0}	p_{f1}

If a link failure means no communication between the two units, then $p_{e0} = 0$ (and thus, $p_{e1} = 1$), because a good tester doesn't produce the *good* test result if it cannot reach the tested unit. But if a link failure means that noise is added to the signal during transmission through the link, then these additional probabilities can have arbitrary values according to the characteristics of the noise.

Fig. 5.3 shows the corresponding modified P-graph.

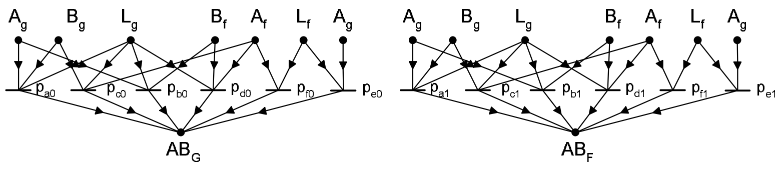


Fig. 5.3. *P-graph model of a single test with potential link error*

5.3. Modeling Comparator-Based Diagnostics. As mentioned above, in the comparison model pairs of units perform the same test and the outcomes are compared. The test result is 0 if they agree, and 1 otherwise (Fig. 5.4).

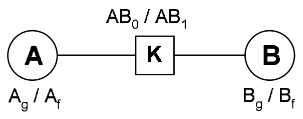


Fig. 5.4. *Fault model of a comparator based test*

5.3.1. Model with Fault-Free Comparators. Traditional models assumes that comparators are fault-free, and according to the behavior of the faulty units and the comparators different diagnostic models were composed. In Malek's model [17] the comparison outcome is 0 only if both units being compared are fault-free. The model introduced by Chwa and Hakimi [18] allows arbitrary comparison outcomes when both the compared units are faulty (Table 5.2).

TABLE 5.2
Traditional test invalidation model of comparator based diagnostic models

State of A	State of B	Test result: Malek's model	Test result: model of Chwa & Hakimi
good	*good*	0	0
good	*faulty*	1	1
faulty	*good*	1	1
faulty	*faulty*	1	x

To be able to handle these models together, parameters are introduced in the test invalidation model for those state combinations, where test outcomes are not exactly determined. This is the case only if both units are faulty (Table 5.3). Parameters represent the probabilities of the 0/1 test outcomes. The application of probabilities means not only the unified handling of traditional models, but it allows to create a more realistic description of the behavior of the system.

TABLE 5.3
Generalized test invalidation model of comparator based diagnostic systems

State of A	State of B	Probability of test result: AB_0	Probability of test result: AB_1
good	*good*	1	0
good	*faulty*	0	1
faulty	*good*	0	1
faulty	*faulty*	p_{d0}	p_{d1}

As it can be seen in Table 5.3, this model arise as a special case of the generalized test invalidation model of the tester–tested unit approach. Hence, the corresponding P-graph appear to be the subgraph of that (the relations with 0 probability are eliminated). Fig. 5.5 shows the P-graph model of a single comparison test.

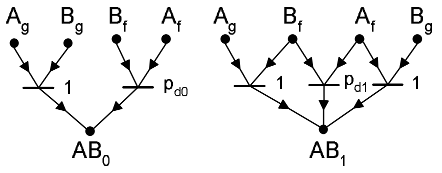


FIG. 5.5. *P-graph model of a single comparator test*

5.3.2. Model with Comparator Errors. The simple model can be extended in order to take into account the potential errors of the comparators. For simplicity, states of the comparators are assumed to be binary (*good* or *faulty*), see Fig. 5.6. According to the new component, the generalized test invalidation table is doubled and probabilities are assigned to both test results for all state combinations (Table 5.4).

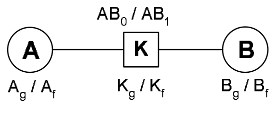


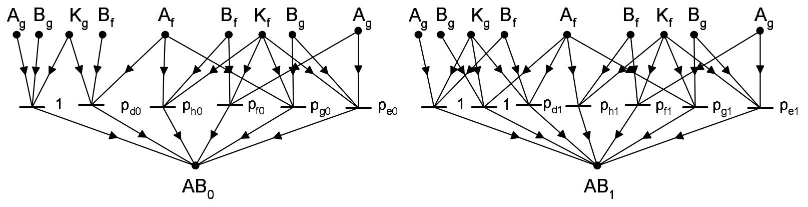
FIG. 5.6. *Fault model of a comparator based test with potential link error*

Following the conversion rule of the invalidation table of a test into a P-graph model, the premisses of the relations are the state combinations, the consequences are the possible test results and the occurring probabilities in the table are assigned to the relations (Fig. 5.7).

The difficulty in creating the model is the determination of the conditional probabilities (for instance the probability of the 0 test result, if both units and the comparator are faulty). It is easier to examine the behavior of the comparator in itself and to derive the searched probabilities from it. Therefore the p_{C00}, p_{C01}, p_{C10}, p_{C11} parameters are introduced, where p_{Cxy} is the probability that a faulty comparator alters the result from x to y. Table 5.4 contains the derivation of the searched probabilities from these parameters, too.

TABLE 5.4
Probabilities of test results in comparator based diagnostic model considering comparator errors

State of comp.	**State of A**	**State of B**	**Probability of test result**	
			AB_0	AB_1
good	*good*	*good*	1	0
good	*good*	*faulty*	0	1
good	*faulty*	*good*	0	1
good	*faulty*	*faulty*	p_{d0}	p_{d1}
faulty	*good*	*good*	$p_{e0} = p_{C00}$	$p_{e1} = p_{C01}$
faulty	*good*	*faulty*	$p_{f0} = p_{C10}$	$p_{f1} = p_{C11}$
faulty	*faulty*	*good*	$p_{g0} = p_{C10}$	$p_{g1} = p_{C11}$
faulty	*faulty*	*faulty*	$p_{h0} = p_{d0}p_{C00} + p_{d1}p_{C10}$	$p_{h1} = p_{d0}p_{C01} + p_{d1}p_{C11}$



A_g B_g K_g B_f A_f B_f K_f B_g A_g
1 p_d0 p_h0 p_f0 p_g0 p_e0
AB_0
A_g B_g K_g B_f A_f B_f K_f B_g A_g
1 1 p_d1 p_h1 p_f1 p_g1 p_e1
AB_1

FIG. 5.7. *P-graph model of a single comparator test with potential comparator error*

6. Example. Consider the test graph and syndrome given on Fig. 2.1 and the test invalidation model given in Table 6.1. Using the base modeling method the initial P-graph contains 6x7 logical relations where probabilities are assigned only to relations. After step 1 in the solution algorithm, the relevant structure contains the half of it, 3x4+3x3 pieces (see Fig. 6.1 without probabilities assigned to state information).

TABLE 6.1
Test invalidation model of the example

State of tester	**State of UUT**	**Probability of test result**	
		0	**1**
good	*good*	1	0
good	*faulty*	0.1	0.9
faulty	*good*	0.7	0.3
faulty	*faulty*	0.4	0.6

During steps 2-4 at most eight solution structure can be generated because of the constraints. Each of it contains six logical relations. The eight structures correspond to the 2^3 possible fault patterns of the three units. That fault pattern is selected finally which produces the syndrome with the highest probability.

If the unit failure probabilities are known (Table 6.2), they can be assigned to the corresponding state information (Fig. 6.1). The 'only' difference to the previously described solution is that the P_S values of subgraphs are different, but this is an important one. Actually it means that the difference between the conditional probabilities of the syndrome for different fault patterns is significantly larger. This results that the solution found at first is closer to the optimum and bigger branches can be bounded.

It can be observed in the first two rows of Table 6.3. The first five columns contain the P_S values of the five fault pattern which is consistent with the syndrome (the biggest value is boldfaced and the second biggest is in italic). The search tree contains 34 nodes if all five solution structure is determined. The sixth column contains the number of nodes which was accessed during the search. It decreased from 17 to 10 when unit failure probabilities were added to the model. The third and fourth rows contain these values for the case when the result of the *test BA* was changed from BA_G to BA_F. In this case the difference is more significant between the two approaches.

If we build into the model more information, namely that there exists mutual tests in the system (Fig. 6.2), then the solution algorithm can be fastened further. After step 1 the maximal structure contains only 2x3+1x4 relations (Fig. 6.3) and it decreases the size of the search tree and the number of accessed nodes. The seventh column in Table 6.3 shows this values; from the maximum 16 nodes only 7 nodes were accessed when unit failure probabilities were also considered. Of course, the conditional probabilities are the same as previously.

Table 6.2
Unit failure probabilities in the example

Unit	$P\{good\}$	$P\{faulty\}$
A	0.99	0.01
B	0.7	0.3
C	0.9	0.1

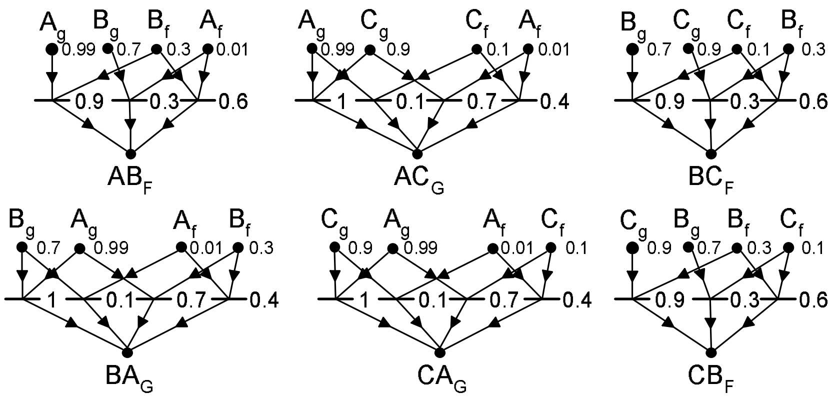


Fig. 6.1. *Maximal P-graph structure of the base model of the example*

Table 6.3
Fault patterns being consistent with syndrome; conditional probabilities of it for base/mutual model with and without unit failure probabilities (FP) and with BA_G/BA_F test result; size of the entire search tree (S) and the number of accessed nodes (n)

		$A_g\ C_g$ B_f	A_g $B_f\ C_f$	B_g $A_f\ C_f$	C_g $A_f\ B_f$	$A_fB_fC_f$	**base (n/S)**	**mutual (n/S)**
BA_G / AB_{FG}	**no FP**	**0.170**	*0.016*	0.001	0.005	0.014	17/34	9/16
	FP	**0.045**	$5*10^{-4}$	$9*10^{-7}$	$1*10^{-5}$	$4*10^{-6}$	10/34	7/16
BA_F / AB_{FF}	**no FP**	**0.073**	0.007	0.011	0.007	*0.021*	25/34	11/16
	FP	**0.019**	$2*10^{-4}$	$8*10^{-6}$	$2*10^{-5}$	$6*10^{-6}$	10/34	7/16

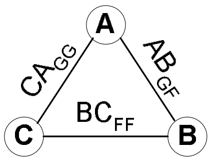


Fig. 6.2. *Test graph of the example containing mutual tests*

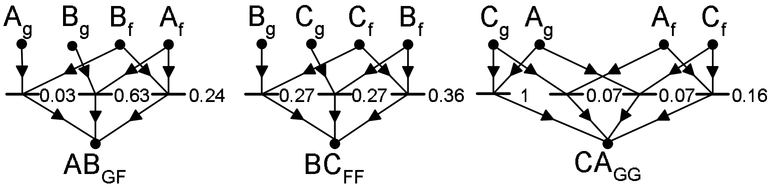


Fig. 6.3. *Maximal P-graph structure of the mutual model of the example*

7. Simulation Results. In order to measure the efficiency of the P-graph based modeling technique a simulation environment was developed, which generates the fault pattern and the corresponding syndrome for the most common topologies with various parameters. The P-graph model of the syndrome-decoding problem was solved as a linear programming task using a commercial program called CPLEX. Other diagnostic algorithms with different solution methods taken from the literature were also implemented for comparison. First, the accuracy of the developed algorithm is demonstrated for varying parameters, then its relation to other algorithms for fixed parameters.

The simulations were performed in a two-dimensional toroidal mesh topology, where each unit is tested by its four neighbors and each unit behaved according to the PMC test invalidation model. Statistical values were calculated on the basis of 100 diagnostic rounds. In every round the fault pattern was generated by setting each processor to be faulty with a given probability, independently from others.

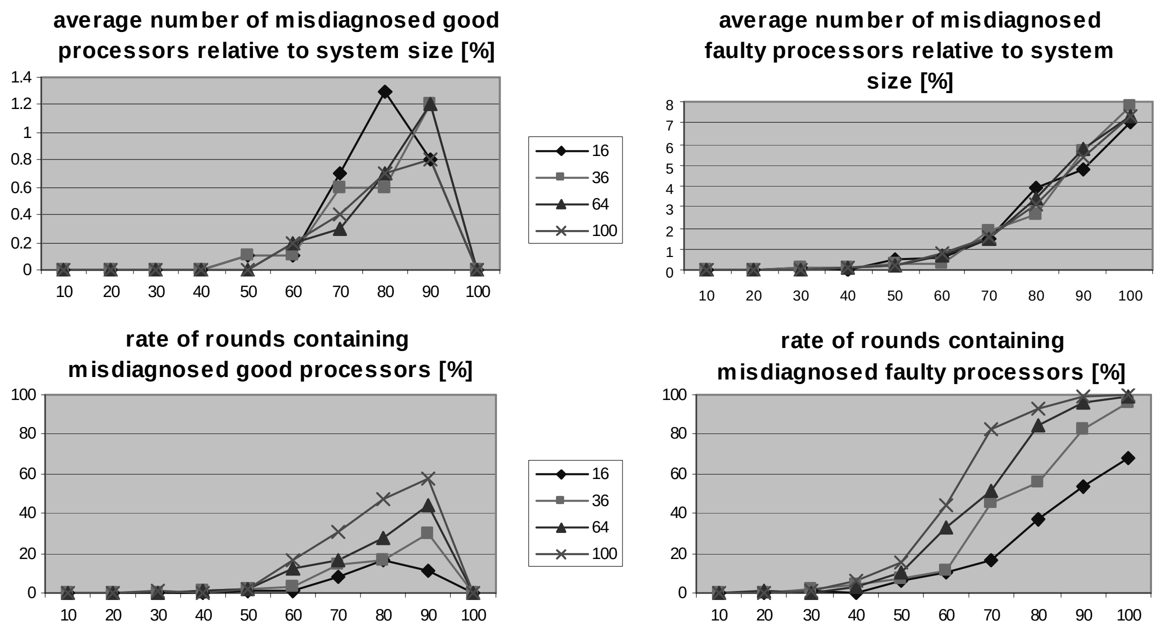


FIG. 7.1. *Simulation results depending on unit failure probability*

Accuracy of the solution algorithm: measurements were performed with system sizes of 4×4, 6×6, 8×8, 10×10 units, and the failure probability of units varied from 10% to 100% in 10% steps. On the diagrams in Figure 7.1 it can be observed that the algorithm has a very good diagnostic accuracy. Even if half of the units were faulty, good units were almost always diagnosed correctly. It is crucial in wafer scale testing because it means that none of good units are thrown away before packaging. Taking still the case when half of the units were faulty the rate of rounds containing misdiagnosed faulty units did not exceed 15%, and the rate of misdiagnosed units relative to the system size was under 1%.

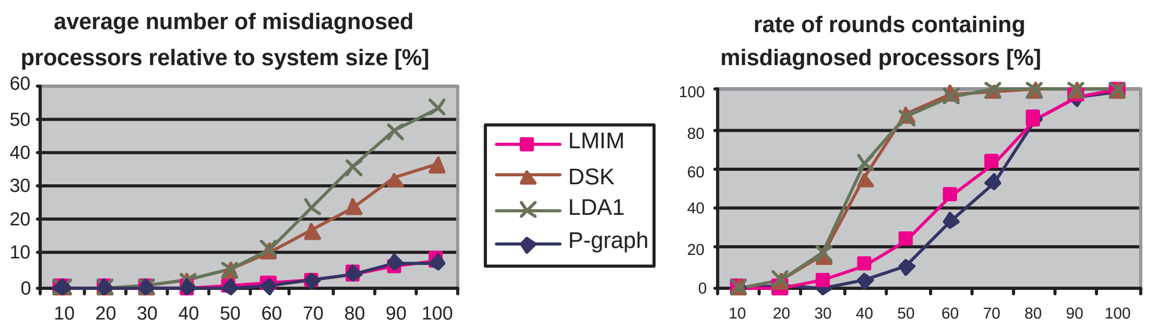


FIG. 7.2. *Comparison of probabilistic diagnostic algorithms*

Comparison to other algorithms: measurements were performed with system size 8×8 and the unit failure probability varied from 10% to 100% in 10% steps. The well-known algorithms taken from the literature were the LDA1 algorithm of Somani and Agarwal [19], the Dahbura, Sabnani and King (DSK) algorithm [20], and the limited multiplication of inference matrix (LMIM) algorithm developed by Bartha and Selenyi [4] from the area of local information diagnosis. It can be seen on the diagrams in Figure 7.2 that only the LMIM-algorithm approximates the accuracy of P-graph-algorithm.

8. Conclusions. Application of P-graph based modeling to system-level diagnosis provides a general framework that supports the solution of several different types of problems, that previously needed numerous different modeling approaches and solution algorithms. The representational power of the model was illustrated in this paper via some practical examples.

Another advantage of the P-graph models is that it takes into consideration more properties of the real system than previous diagnostic models. Therefore its diagnostic accuracy is also better. This means that it provides almost good diagnosis even when half of the processors are faulty, which is important in the field of wafer scale testing.

The favorable properties of the approach are achieved by considering the diagnostic system as a structured set of hypotheses with well-defined relations. The syndrome-decoding problem in multiprocessor systems has a special structure, namely the direct manifestation of internal fault states in the syndromes. In more complex systems the states of the control logic have to be taken into account in the model to be analyzed [21]. These straightforward extensions to the modeling of integrated diagnostics can be well incorporated into the P-graph based models. Our current work aims at generalization of the results into this direction by extending previous results on the qualitative modeling of dependable systems with quantitative optimization [22].

Acknowledgments. This research was supported by the Hungarian National Science Fund grant OTKA T032408, which is gratefully acknowledged by the authors.

REFERENCES

[1] T. Bartha, E. Selényi, *Probabilistic System-Level Fault Diagnostic Algorithms for Multiprocessors*, Parallel Computing, vol. 22, no. 13, pp. 1807–1821, Elsevier Science, 1997.

[2] T. Bartha, *Efficient System-Level Fault Diagnosis of Large Multiprocessor Systems*, Ph.D thesis, BME-MIT, 2000.

[3] M. Barborak, M. Malek, and A. Dahbura, *The Consensus Problem in Fault Tolerant Computing*, ACM Computing Surveys, vol. 25, pp. 171–220, June 1993.

[4] T. Bartha, E. Selényi, *Probabilistic Fault Diagnosis in Large, Heterogeneous Computing Systems*, Periodica Polytechnica, vol. 43/2, pp. 127–149, 2000.

[5] B. Polgár, Sz. Nováki, A. Pataricza, F. Friedler, *A Process-Graph Based Formulation of the Syndrome-Decoding Problem*, DDECS 2001, 4th Workshop on Design and Diagnostics of Electronic Circuits and Systems, pp. 267–272, Hungary, 2001.

[6] S. N. Maheshwari, S. L. Hakimi, *On Models for Diagnosable Systems and Probabilistic Fault Diagnosis*, IEEE Transactions on Computers, vol. C-25, pp. 228–236, 1976.

[7] B. Polgár, T. Bartha, E. Selényi, *Modeling Uncertainty In System-Level Fault Diagnosis Using Process Graphs*, DAPSYS 2002, 4th Austrian-Hungarian Workshop on Distributed and Parallel Systems, pp. 195–200, Linz, Austria, Sept 29–Okt 2, 2002.

[8] M. L. Blount, *Probabilistic Treatment of Diagnosis in Digital Systems*, in Proc. of 7th IEEE International Symposium on Fault-Tolerant Computing (FTCS-7), pp. 72–77, June 1977.

[9] F. Friedler, K. Tarjan, Y. W. Huang, L. T. Fan, *Combinatorial Algorithms for Process Synthesis*, Comp. in Chemical Engineering, vol. 16, pp. 313–320, 1992.

[10] F. Friedler, K. Tarjan, Y. W. Huang, and L. T. Fan, *Graph-Theoretic Approach to Process Synthesis: Axioms and Theorems*, Chemical Engineering Science, 47(8), pp. 1973–1988, 1992.

[11] F. Friedler, L. T. Fan, and B. Imreh, *Process Network Synthesis: Problem Definition*, Networks, 28(2), pp. 119–124, 1998.

[12] F. Aglietti, et al., *Self-Diagnosis of APEmille*, Proc. EDCC-2 Companion Workshop on Dependable Computing, pp. 73–84, Silesian Technical University, Gliwice Poland, May 1996.

[13] A. Petri, P. Urbán, J. Altmann, M. Dal Cin, E. Selényi, K. Tilly, A. Pataricza, *Constraint-Based Diagnosis Algorithms for Multiprocessors*, Periodica Polytechnica Ser. El. Eng., Vol. 40, No. 1, (1996), pp. 39-52.

[14] S. Lee and K. G. Shin,*Optimal multiple syndrome diagnosis*, Digest of Papers, FTCS-20, Newcastle Upon Tyne, United Kingdom, pp. 324–331, June 1990.

[15] B. Sallay, P. Maestrini, P. Santi, *ComparisonBased Wafer-Scale Diagnosis Tolerating Comparator Faults*, IEEE Journal on Computers and Digital Techniques, 146(4), pp. 211–215, 1999.

[16] S. Chessa, *Self-Diagnosis of Grid-Interconnected Systems, with Application to Self-Test of VLSI Wafers*, Ph.D. Thesis, TD-2/99 University of Pisa, Italy, March 1999.

[17] M. Malek, *A comparison connection assigment for diagnosis of multiprocessor systems*, Proc. 7th FTCS, pp. 31-35, May 1980.

[18] K.Y. Chwa, S.L. Hakimi, *Schemes for fault tolerant computing: a comparison of modulary redundant and tdiagnosable systems*, Information and Controls, vol. 45, No. 3, pp. 212-238, 1981.

[19] A. Somani, V. Agarwal, *Distributed syndrome decoding for regular interconnected structures*, in 19th IEEE International Symposium on Fault Tolerant Computing, pp. 70–77, IEEE 1989.

[20] A. Dahbura, K. Sabnani, and L. King, *The Comparison Approach to Multiprocessor Fault Diagnosis*, IEEE Transactions on Computers, vol. C-36, pp. 373–378, Mar. 1987.
[21] A. Pataricza, *Algebraic Modelling of Diagnostic Problems in HW-SW Co-Design*, in Digest of Abstracts of the IEEE International Workshop on Embedded Fault-Tolerant Systems, Dallas, Texas, Sept. 1996.
[22] A. Pataricza, *Semi-decisions in the validation of dependable systems*, Proc. of IEEE International Conference on Dependable Systems and Networks, pp. 114–115, Göteborg, Sweden, July 2001.

Edited by: Zsolt Nemeth, Dieter Kranzlmuller, Peter Kacsuk, Jens Volkert
Received: April 1, 2003
Accepted: June 5, 2003

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 59–70. http://www.scpe.org

ISSN 1895-1767
© 2005 SWPS

TOLERATING STOP FAILURES IN DISTRIBUTED MAPLE

KÁROLY BÓSA[†] AND WOLFGANG SCHREINER[†]

Abstract. In previous work we have introduced some fault tolerance mechanisms to the parallel computer algebra system Distributed Maple such that a session may tolerate the failure of computing nodes and of connections between nodes without overall failure. In this paper, we extend this fault tolerance by some advanced mechanisms. The first one is the reconnection of a node after a connection failure such that a session does not deadlock. The second mechanism is the restarting of a node after a failure such that the session does not fail. The third mechanism is the change of the root node such that a session may tolerate also the failure of the root without overall failure.

Key words. distributed systems, fault tolerance, computer algebra.

1. Introduction. Distributed Maple (see Figure 1.1) is a Java-based system for implementing parallel programs in the computer algebra system Maple [26]. We have employed this system successfully for the parallelization of various methods and applications in algebraic geometry [28, 29]. The system is portable and has been used in numerous parallel and networked environments, e.g. clusters of Linux PCs and Unix workstations, a Sun HPC 6500 bus-based shared memory machine, an Origin 2000 NUMA multiprocessor, and heterogeneous combinations of these. Recently we have analyzed the system performance in these environments in large detail [27].

We have used the system to develop the parallel versions for various non-trivial applications from computer algebra and algebraic geometry, e.g. bivariate resultant computation, real root isolation, plotting of algebraic plane curves, plotting of surface to surface intersections and neighborhood analysis of algebraic curves. A comprehensive survey on the system and its applications is given in [31].

Distributed Maple has evolved from our own experience in the development of parallel computer algebra environments and from learning from the work of other researchers. The programming interface of the system is based on a para-functional model as adopted by the PARSAC-2 system [16] for computer algebra on a shared memory multiprocessor. The model was refined in PACLIB [11] on which a para-functional language was based [25].

The only mechanism originally available in Distributed Maple for dealing with faults was the watchdog mechanism for shutting down the system in case of failures. However, as we began to attack larger and larger problems, the meantime between session failures (less than a day) became a limiting factor in the applicability of the system.

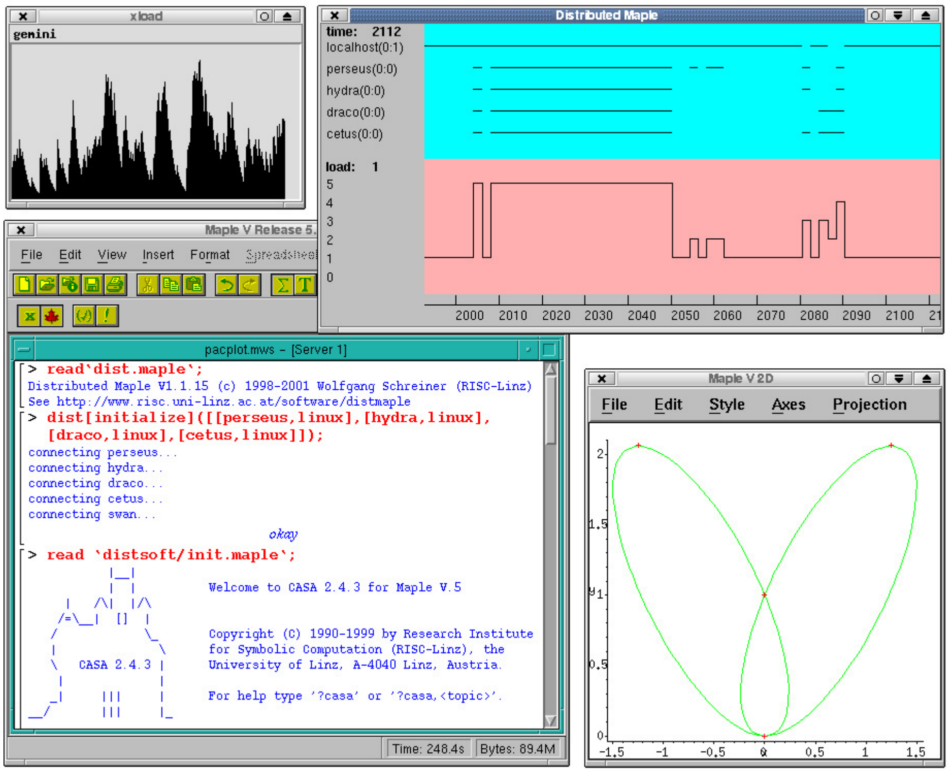
There are numerous possibilities for faults that may cause a session failure: a machine becomes unreachable (usually a transient fault, i.e., the machine is rebooted or is temporarily disconnected), a process in the scheduling layer or in the computation layer crashes (a bug in the implementation of the Java Virtual Machine, of the Java Development Kit, of Maple, or of Distributed Maple itself) or the computation itself aborts unexpectedly (a bug in the application). While the last issue can be overcome and the Distributed Maple software itself is very stable, there certainly exist bugs in the lower software levels that are out of our control; machine/network/operating system faults may happen in any case.

We have therefore started to investigate how to introduce fault tolerance mechanisms that let the system deal with faults in such a way that the time spent in long running session is not wasted by an eventual failure. There exist various fundamental mechanisms to achieve fault tolerance in distributed systems [5, 13]:

(i) By *replication* we understand the duplication of resources or services on multiple servers such that, if a server fails, a request is handled by another one; because parallel programming systems are concerned about optimal use of resources, only few systems apply this approach [1, 2].

(ii) Most frequently, parallel programming environments pursue fault tolerance by *checkpointing*: they regularly save global snapshots of the session state on stable storage such that a failed session can be re-started from the last saved state (potentially migrating tasks to other machines); stable storage can be implemented by disks or by replicated copies in the memories of multiple machines [22, 24]. Some systems perform checkpointing

[†] *Research Institute for Symbolic Computation (RISC-Linz), Johannes Kepler University, Linz, Austria*, http://www.risc.uni-linz.ac.at, e-mail: FirstName.LastName@risc.uni-linz.ac.at.



```
> read`dist.maple`;
Distributed Maple V1.1.15 (c) 1998-2001 Wolfgang Schreiner (RISC-Linz)
See http://www.risc.uni-linz.ac.at/software/distmaple
> dist[initialize]([[perseus,linux],[hydra,linux],
  [draco,linux],[cetus,linux]]);
connecting perseus...
connecting hydra...
connecting draco...
connecting cetus...
connecting swan...
                                 okay
> read `distsoft/init.maple`;
           |__|
           |  |         Welcome to CASA 2.4.3 for Maple V.5
       |   /\|  |/\
      /=\__|  []  |
     /            \_     Copyright (C) 1990-1999 by Research Institute
     |              \    for Symbolic Computation (RISC-Linz), the
     \   CASA 2.4.3 |    University of Linz, A-4040 Linz, Austria.
      |             |
     _|    |||      |    For help type '?casa' or '?casa,<topic>'.
  __/      |||      |_
```

Fig. 1.1. *The User Interface of Distributed Maple*

transparently to the application, often on top of PVM [7, 6, 10, 17, 15] or MPI [23]. Other systems rely on application support for checkpointing [4, 21].

(iii) Many metacomputing and message passing frameworks do not provide built-in fault tolerance at all. Rather they include failure detection and handling services that enable the user to deal with failures at the application-level. Examples of such systems are the Globus toolkit [32], the Harness environment [18], PVM [9] or FT-MPI [8].

All these approaches are relatively complex because they have to deal with general parallel computations; for special problems much simpler solutions exist [12]. However, also parallel programming models that are more abstract than message passing should allow to deal with fault tolerance in a simpler way. While this is not yet completely true for the coordination language Linda [3], the *functional* programming model provides this potential to a large degree [14].

All of the possibly fault cases mentioned above can be classified into the following failure types: *Message* failures, *Stop* failures or *Byzantine* failures [19]. We do not deal with tolerating of *Message* failures and *Byzantine* failures (our system currently is not able to handle corrupt messages and corrupt task executions), but in the functional parallel programming model, the handling of these two kinds of failure situations may happen similarly as in the message passing or any other general parallel programming model (there are no simpler solutions for these in the functional model). Furthermore, there exist already numerous general and effective algorithms for tolerating of these kinds of failures [19].

We focus on *Stop* failures and we assume in this paper that only such failures may occur (a part of the system can exhibit *Stop* failure simply by stopping somewhere in the middle of its execution). In more detail, we deal with the following error situations: the root of the tree of computation nodes crashes, some non-root node crashes, some Maple process fails, a connection between the root and another node breaks and a connection between two non-root nodes breaks. We say that a node crashes if the machine stops execution or reboots, or if the Distributed Maple process aborts.

In the original system, we had to restart the computation in each of these cases. If the root crashes, the system aborts; in all other cases, it deadlocks and must be stopped manually. Our goal was to extend the

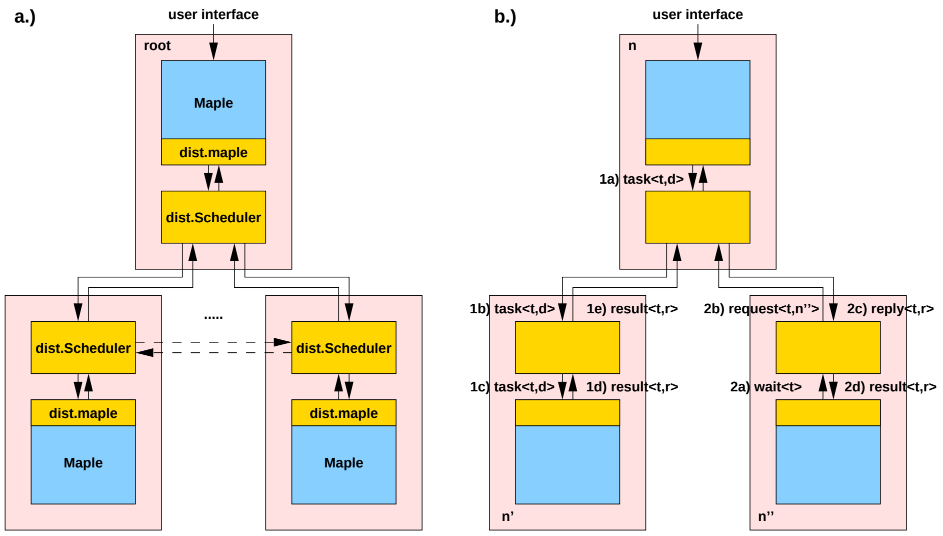


FIG. 2.1. ***a.)*** *System Model* ***b.)*** *Execution Model*

time scale of distributed computations (measured in hours at the moment) to many days as is required for realistic applications. Another requirement was that the implemented fault tolerance has to be *transparent* to the application.

Earlier, we introduced three fault tolerance mechanisms, by which some of the mentioned *Stop* failure problems were covered. The functional task model of Distributed Maple considerably simplified these developments [30].

Now, we have extended these mechanisms to cover more *Stop* failures: first, a session may tolerate node failures (except root failure) and connection failures without loss of resources (e.g.: computed task results, Maple kernels or schedulers); second, we have introduced a mechanism to avoid a session failure, even if the root becomes unreachable for most of the non-root nodes. With these mechanisms, Distributed Maple is by far the most advanced system for computer algebra concerning reliability in distributed environments.

The remainder of this paper is organized as follows: Section 2 introduces the model of the system and its operation. Section 3 gives a short overview about the previously achieved fault tolerance mechanisms on which the following descriptions are based. Section 4 explains the reconnection and restarting mechanisms that we have introduced to reduce the loss of resources after node or connection failures. Section 5 extends this by a mechanism that allows the system to cope with root node failure without overall failure. Section 6 compares the fault tolerance features of Distributed Maple with a checkpointing mechanism developed for a particular parallel programming environment. Section 7 summarizes our results and outlines further development directions.

2. System and Execution Model. A Distributed Maple session comprises a set of *nodes* (machines, processors) each of which holds a pair of processes: a (e.g. Maple) *kernel* which performs the actual computation, and a *scheduler* (a Java Virtual Machine executing a Java program) which distributes and coordinates the tasks created by all kernels (see Figure 2.1a). The scheduler communicates, via pairs of sockets, with the kernel on the same node and, via sockets, with the schedulers on the other nodes. The communication protocol between the scheduler and the kernel is implemented by *dist.maple* interface. The *root* is that node from which the session was established by user interaction with the kernel. Initially, a single task is executing on the root kernel; this task may subsequently create new tasks which are distributed via the schedulers to other kernels and may in turn create new tasks.

The parallel programming model is basically functional (see Figure 2.1b): from the perspective of a scheduler, a kernel may emit a number of messages `task:`$<t, d>$ where t identifies the task and d describes it. The task needs to be forwarded to some idle kernel which eventually returns a message `result:`$<t, r>$ where r represents the computed result. When a kernel emits `wait:`$<t>$, this task is blocked until the scheduler responds with the result of t. Thus, if this result is not yet available, the scheduler may submit to the now idle kernel another task; when this task has returned its result, the kernel may receive

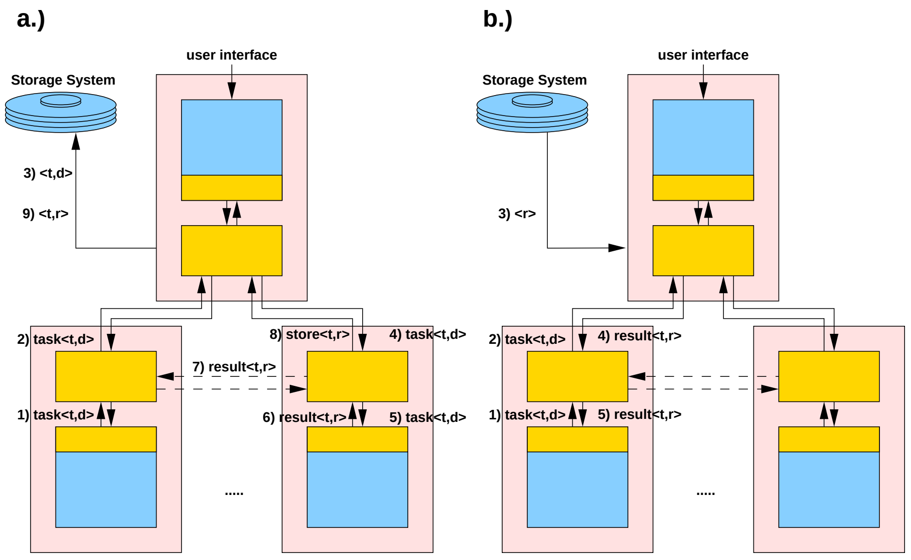


FIG. 3.1. ***a.)*** *Logging a Result* ***b.)*** *Restoring a Result*

the result waited for or yet another task. A task identifier t encodes a pair $<n,\ i>$ where n identifies the node on which the task was created and i is an index. The node n thus serves as the rendezvous point between node *n'* computing the result of t and node *n"* requesting this result. When the scheduler on n receives a `task:`$<t,\ d>$ from its kernel, it thus allocates a result descriptor that will eventually hold the result r; the task itself is scheduled for execution on some node *n'*. When a kernel on some node *n"* issues a `wait:`$<t>$, the scheduler on *n"* forwards a `request:`$<t,\ n''>$ to n. If r is not yet available on n, this request is queued in the result descriptor. When the kernel on *n'* eventually returns the `result:`$<t,\ r>$, the scheduler on *n'* forwards this message to n, which constructs a `reply:`$<t,\ r>$ and sends it to *n"*.

3. Formerly Achieved Fault Tolerance. The only mechanism originally available in Distributed Maple for handling faults was a watchdog mechanism on every non-root node that regularly checked whether a message had been recently received from the root. If not, it sent a `ping` message that had to be acknowledged. If no acknowledgement arrived from the root within a certain time bound, the node shut itself down.

We have extended this basic operation by three fault tolerance mechanisms. For the correct working of these mechanisms, we assume that the system has access to a reliable and stable storage system which never loses data. Communication is reliable, i.e. messages are never corrupted or duplicated (no *Message* failures) and they may be lost only in *Stop* failure situations. No *Byzantine* failures may occur.

Logging Results The logging mechanism can store a consistent state of a session and is able to recover this stored state after a failure [30]. This mechanism is based on the scheduling mechanism originally available in Distributed Maple, which allows the root to receive all task descriptions created by all nodes (see Figure 3.1a). Before the root schedules a `task:`$<t,\ d>$, it saves *d*. When a node sends a `result:`$<t\ ,r>$ to some other node different from the root, it forwards a copy in a `store:`$<t,\ r>$ to the root which saves it. When a session is restarted after a failure (see Figure 3.1b), the root may receive a `task:`$<t',\ d>$. If d is already stored, the root checks whether there exists stored r for the received d and it tries to restore this *r*. The mechanism is able to recognize the corrupted files and it uses only the properly stored data in the recovery phase.

Tolerating Node Failures The system is capable to cope with node failures without overall failure [30]. This mechanism is based on the original watchdog mechanism. It sends a `ping` to a node and supposes its failure, if it does not receive any reply in a limited time period. If the root observes that a node become unreachable, the root declares it as *dead*. The "Tolerating Node Failures" mechanism is based on the "Logging Results" mechanism, because all results that have been stored on the dead node are recovered and provided by the root. The root also recovers and reschedules to another node those task descriptors that were under processing on the dead node.

Tolerating Peer Connection Failures The connection between two non-root nodes is called peer connection. Such connections are established on demand during the execution. The system is capable to cope with peer connection failures without overall failure. This mechanism is based on "Tolerating Node Failures" mechanism, because it assumes that the system is able to continue normal operation, if a connection between root and another node fails. The principle is simple: if a non-root node cannot send a message to another such node, it sends this message to the root which forwards it to the final destination.

The effect of these mechanisms on the performance of the system has not been measured yet, but we do not expect it to be significant. For instance in the case of the "Logging Results" mechanism, task descriptions and task results are saved by separate threads and thus do not hamper the normal flow of operation. These descriptions and results are also communicated in the basic operation model; their size is therefore bound by the performance of the communication system rather than by the extra overhead of the saving mechanism. Additionally, the "Logging Results" mechanism introduces only one extra message type, the `store:`$<t, r>$ message, for saving of the task results to the storage system (task descriptions are saved when tasks are scheduled).

The overhead of the "Tolerating Node Failure" and the "Tolerating Peer Connection Failure" mechanisms is also not significant, because these mechanisms maintain only one small extra data structure on the root node and another one on each non-root node. They do not use additional messages during the normal operation.

4. Reducing the Loss of Resources after *Stop* Failures. In this section, we describe two quite simple extensions of our basic fault tolerance mechanisms by which the system is able to reduce the loss of resources after some kind of *Stop* failures.

4.1. Reconnection after Connection Failure. We have extended the basic fault tolerance mechanisms described in the previous section such that the root tries to recreate the connection to a disconnected node before it starts the migration of tasks in the "Tolerating Node Failures" mechanism.

If a node i observes that the root has become unreachable, it waits for a connection request from the root for a limited time period. If the request does not arrive, i shuts itself down. When a root observes node i becomes unreachable, it tries to reconnect to i in the same way in which a node creates a peer connection to another node during normal execution. If it does not get back any reply, then it starts the "Tolerating Node Failures" mechanism. If i receives a connection request from the root, it sends back a positive acknowledgement (independently whether it has already observed the connection failure or not). When the root gets this request, it informs all other nodes about the connection failure and the successful reconnection to i.

There is a main problem that all nodes (root and all others) now have to deal with: the management of those messages that were sent and might be lost. For solving this, the root can resend some task, result and wait messages to node i, node i can resend some task, result, store and wait messages to the root or some other nodes, and all other nodes can resend some result and wait messages via the root to node i.

For resending the corresponding wait messages, every node can use a set W_j which is used and maintained in Distributed Maple. W_j contains all wait messages sent to j; for a subset W_j^r the answers with results have already arrived from j. After j reconnected, each wait message in $W_j - W_j^r$ has to be sent again.

For resending the corresponding task, result, and store messages, every node maintains another set M_j, which contains all task, result and store messages sent to j (see Figure 4.1a). On the root, this set contains also those result messages which were sent by any other node to j through the root. For a subset M_j^a, acknowledgements have already arrived from j (see Figure 4.1b). For acknowledging these messages, the system uses the modified ping message and its acknowledgement. Namely, if the number of non-acknowledged messages reaches a certain bound in M_j, a `ping:`$<k>$ message is sent and the set $M_{j,k}$ becomes $M_j - M_j^a$ where k uniquely identifies this `ping:`$<k>$ message (such a message is also sent in the original case, see Section 3). If an acknowledgement arrives with index k, every element of $M_{j,k}$ is added to M_j^a. After j has been reconnected, each message in $M_j - M_j^a$ has to be sent again (see Figure 4.1c and d).

It may occur that some message arrives twice. This is not a problem, because the system tolerates redundant messages.

4.2. Restarting after Node Failure. We have implemented a quite simple extension of the "Tolerating Node Failures" mechanism by which the root tries to restart the crashed or aborted nodes: after the unsuccessful reconnection to node i, the "Tolerating Node Failures" mechanism is started. The root also starts an asynchronous thread to try to restart eventually i in the same way as in initial phase. If this is managed, node i

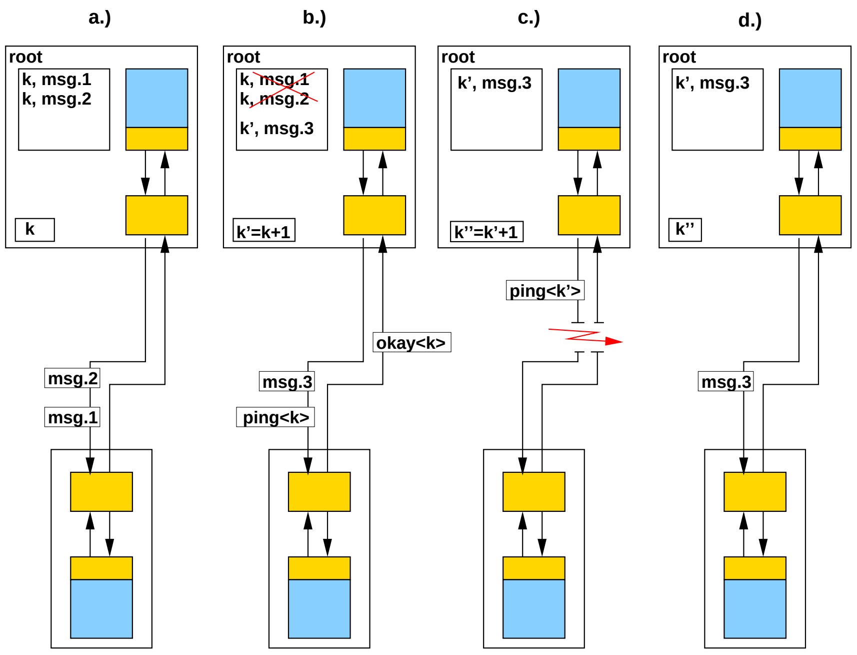


FIG. 4.1. *The resending method:* ***a.)*** *Sent messages are stored in a table.* ***b.)*** *If the acknowledgement message has arrived for the* ping *message, the corresponding messages are removed from the table.* ***c.)*** *The connection fails. The acknowledgement message for the ping does not arrive within a certain time bound.* ***d.)*** *After the reconnection, the messages in the table are resent.*

gets a new identifier instead of i, because all the results that have been stored on i earlier are provided by the root during the rest of the execution. The targets of the wait messages have to be determined uniquely from the node identifier contained in the task identifier.

By changing the identifier of the restarted node, we can guarantee that all other nodes interact to the newly started node. Namely, if node i did not fail, just disconnected to the root, it may send some messages to some other nodes. But in this case, these nodes simply drop these messages (because i is declared dead by “Tolerating Node Failures” mechanism). i eventually aborts.

5. Tolerating Root Failure. This mechanism is based on all already mentioned fault tolerance mechanisms (Logging Results, Tolerating Node Failure, Tolerating Peer Connection Failures and Reconnection after Connection Failure) except ”Restarting after Node Failure”. Therefore, the same assumptions are made as were described for the previous mechanisms: a reliable and stable storage system, reliable communication, no *Byzantine* failures. There are two more important assumptions. First, the storage system is independent from the root and at least one non-root node has access to it. Second, the description of the main task which is performed by the root kernel is initially stored by this storage system.

At no time during the execution of the system, there may exist more than one root. If the root becomes unreachable to another node, this means either the root crashed (see Figure 5.1a) or the connection between the root and the node broke (see Figure 5.1b, c and d). In the second case, the system has to be able to decide about the election of a new root. To guarantee this, the current root always has to be connected to $n/2$ nodes at least (see Figure 5.1c); a non-root node may become the root if and only if at least $n/2$ non-root nodes (beyond itself) accept it as the root (see Figure 5.1d), where n is the initial number of the non-root nodes.

It is possible to use the ”Restarting after Node Failure” mechanism together with this mechanism, but the additional restriction is needed: an unreachable node may be restarted if and only if the root has declared it dead (see Section 3) and more than $n/2$ nodes have acknowledged this declaration (the ”Tolerating Node Failures” mechanism warrants that every message from a dead node is dropped).

At the initialization of the session, all nodes get the identifier of a special non-root node which can access the storage system. This node is called shadow root or simply shadow. The subsequent explanations assume that there is only one such shadow. In Section 5.5, we will generalize the mechanism to an arbitrary number of (potential) shadows.

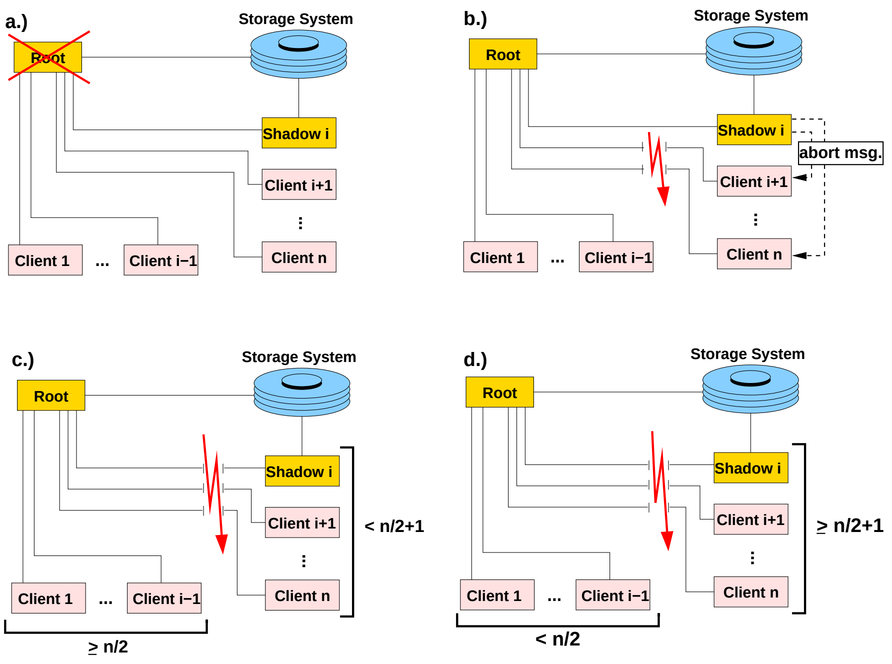


FIG. 5.1. ***a.)*** *The root crashes.* ***b.)*** *The connections between the root and some non-shadow nodes break.* ***c.)*** *The network is split. The root is connected to at least $n/2$ pieces of non-root nodes.* ***d.)*** *The network is split. The shadow node is connected to at least $n/2$ pieces of non-root nodes.*

5.1. Triggering the Root Election. If the root becomes unreachable to a non-shadow node k and the reconnection time expires, k sends a `root_lost` message directly to the shadow node i. If node i has become also unreachable, k aborts. Shadow i maintains a set R of the identifiers of all nodes that cannot contact the root. When i receives the `root_lost` message from k, it adds k to R. From the moment that the first node is added to R, i waits for a certain period of time. If during this period, no message arrives from the root, i starts the root election. However, if during this time a message from the root arrives (i.e. the root is not unreachable to i), the election is canceled (see Figure 5.1b). In this case, i waits until the root declares k dead (which must eventually happen since k cannot contact the root), and then it sends an `abort` message to k (which causes k to abort, but the root will eventually restart k). Summarizing, root election is only started in the situations illustrated in Figure 5.1a, c and d.

5.2. Performing the Root Election. Shadow node i broadcasts a `check_root` message to all live nodes except those whose identifiers are in R. When a node l receives such a message, it sends back an `acknowledgement` message. Node l also checks the root within a certain time bound. If the root is reachable to l, then l sends back a `root_connected` message to the shadow node i. Otherwise, it sends back a `root_lost` message to i. Node i waits for the `acknowledgement` messages of the `check_root` broadcast for a limited time period and counts them. If this sum plus the size of R is less than $n/2$ where n is the initial number of the non-root nodes, i aborts (see Figure 5.1c). Otherwise, it waits further for `root_lost` and `root_connected` messages and counts them, too. If it receives a `root_lost` message, it adds the identifier of the sender to R (if i observes that a node whose identifier is in R became unreachable, i deletes the corresponding identifier from R). If the number of `root_lost` messages reaches the bound $n/2$, i sends a `new_root` message to all other nodes. If this bound has not been reached, but a `root_lost` or a `root_connected` message has been received from each acknowledged node that is reachable to i, i aborts.

Each node that has received the `new_root` message accepts i as the new root even if the old root is still reachable to it. Summarizing, the shadow node i becomes the new root only in cases represented in Figure 5.1a and d.

In case depicted in Figure 5.1d, the old root eventually realizes that less than $n/2$ nodes are connected and aborts.

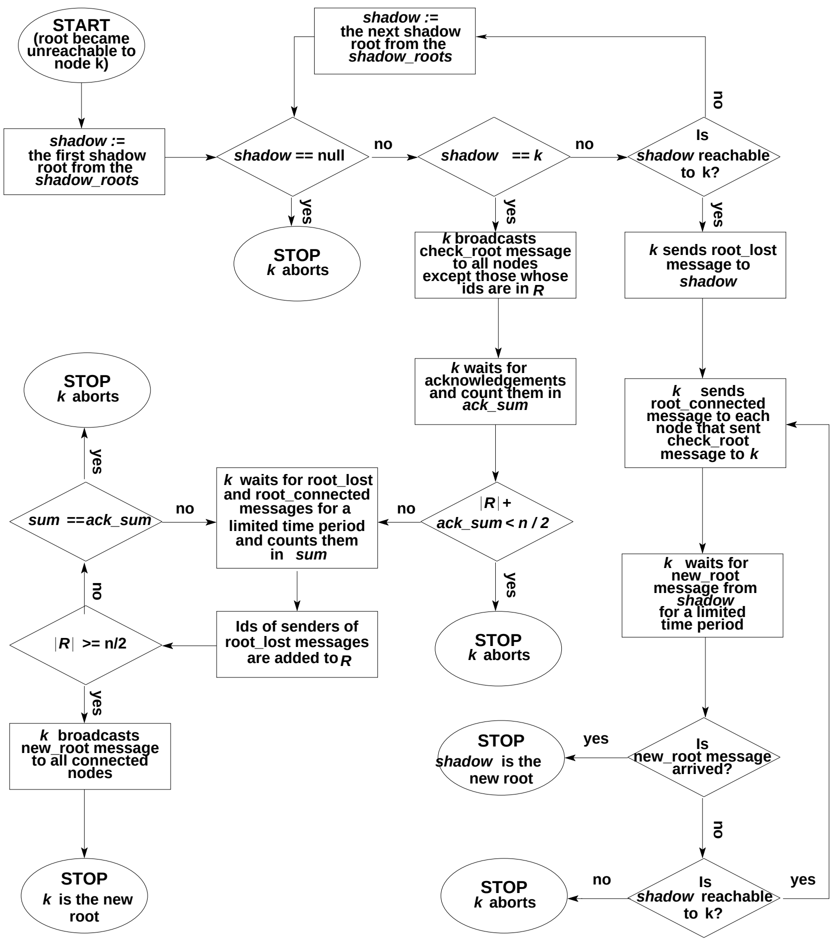

START (root became unreachable to node k)
shadow := the first shadow root from the shadow_roots
shadow == null
yes
no
STOP k aborts
shadow := the next shadow root from the shadow_roots
shadow == k
yes
no
Is shadow reachable to k?
yes
no
k broadcasts check_root message to all nodes except those whose ids are in R
k waits for acknowledgements and count them in ack_sum
|R| + ack_sum < n / 2
yes
no
STOP k aborts
k waits for root_lost and root_connected messages for a limited time period and counts them in sum
Ids of senders of root_lost messages are added to R
|R| >= n/2
yes
no
sum ==ack_sum
yes
no
STOP k aborts
k broadcasts new_root message to all connected nodes
STOP k is the new root
k sends root_lost message to shadow
k sends root_connected message to each node that sent check_root message to k
k waits for new_root message from shadow for a limited time period
Is new_root message arrived?
yes
no
STOP shadow is the new root
Is shadow reachable to k?
yes
no
STOP k aborts

Fig. 5.2. *The generalized root election method on node* k *where* n *is the initial number of the non_root nodes.* R *is a set which consists of the identifiers of those nodes that sent root_lost message to* k. |R| *signs the size of* R.

5.3. Initialization of the New Root. After the shadow root has become the root, it loads the main task from the storage system and schedules it to its own kernel. Then it declares the old root and those nodes dead which did not acknowledge the receipt of the `check_root` message (when the connected nodes receive this declaration, they resend those wait messages to the new root which might be lost, see Section 3). After a node has accepted a `new_root` message, it resends all store and task messages to the new root which are not acknowledged by the old root (task messages are acknowledged by the root if and only if they have already been logged). It also sends the identifiers of those tasks in a `scheduled_tasks:`<*task_identifiers*> message to the new root which are under processing on this node. The new root keeps these pieces of information in the *scheduled_tasks* table as tuples of a node identifier and a task identifier (this table is the same which are already used by the root in the "Tolerating Node Failures" mechanism in Section 3).

5.4. Task Handling and Termination. If the new root receives a `task:`<*t, d*> message, the logging mechanism tries to restore the result of the task from *d*. If it does not manage, it checks whether *d* may be already logged with a different identifier *t'*. If yes, it checks whether *t'* occurs in the *scheduled_tasks* table. If *t'* occurs in this table, the new root drops this message, because it is already under processing somewhere. Otherwise, it schedules this task to a node.

In the normal case, when there is no root failure during the execution, the termination of the system is always triggered by user interaction. But now that the root of the session is changed, this is not possible any more (because the user interface of the system is located on the initial root, see Figure 2.1a). Therefore, after the shadow root has become the root, it counts separately how many task descriptors and task results are already stored by "Logging Results" mechanism and maintains these pieces of information. If the number of

	P–GRADE checkpointing	**fault tolerance in Distributed Maple**
architecture	centralized	centralized
applicability	it is targeted to general parallel applications	it is restricted to parallel programming models based on functional tasks
transparency	transparent	transparent
the saving method	a computing phase is periodically interrupted by a checkpointing phase	continuous
if the system fails during the computing phase	it can be restarted from the last checkpoint	it is able to recognize the corruption of a file; thus it may reuse each properly stored datum in a later session
if the system fails during the check–pointing phase	it can be restarted only from the previous checkpoint	
in case of node failure	it is able to continue execution from the last checkpoint (except if the server fails)	it is able to continue execution by the migration of the tasks (also if the root fails)

Fig. 6.1. *Comparison of Fault Tolerance in Distributed Maple and P-GRADE Checkpointing*

stored task results becomes equal to the number of stored task descriptors, the system terminates (the “Logging Results” mechanism warrants that the descriptor of a task is always stored before its own result and the result of its parent task). In a later session, the system is able to recover the result of the whole computation by the “Logging Results” mechanism.

5.5. Generalization of the Root Election. Above description uses only one shadow node. Now, we generalize the mechanism such that we use a list of the shadow nodes. This list is called *shadow_roots*; all nodes receive it at the initialization of the session.

If the root is unreachable to a node k and the reconnection time expires but the first node in the *shadow_roots* has become also unreachable, k sends a `root_lost` message to the first live node in this list (see Figure 5.2). If such a node does not exist, k aborts. If k has already sent a `root_lost` message to a node and some `check_root` messages arrive from some other nodes, it replies with a `root_connected` messages. If k is the next shadow root, it broadcasts a `check_root` message as described in the previous section.

Finally, the number of the elements of R on each shadow root decides the result of the root election. In the worst case, neither shadow node becomes the root and the whole system aborts.

6. Comparison with a Particular Checkpointing Mechanism. In order to put our work in context, we are now going to compare the fault tolerance features of Distributed Maple with those of systems based on checkpointing mechanisms. As a concrete example, we take the P-GRADE environment into which such a mechanism has been recently integrated [15] (see Figure 6.1). P-GRADE is a parallel programming environment which supports the whole life-cycle of parallel program development [20]. A P-GRADE application is always centralized, where a server coordinates the start-up phase and the execution. The primary goal of the P-GRADE checkpointing mechanism was to make P-GRADE generated applications able to migrate among nodes within a cluster. Its secondary goal was to make periodic checkpoint for supporting fault tolerance.

In the case of the P-GRADE checkpointing, the main idea was to save the states of the processes by making regularly snapshots of them. Thus during the execution a computing phase periodically alternates with a checkpointing phase. As mentioned in Section 1, such an approach is relatively complex because it is targeted to general parallel applications. In the case of our “Logging Results” mechanism, the system saves the computed task results instead of the states of the processes and this saving operation is continuous. This solution is simpler, but it is restricted to parallel programming models based on functional tasks, i.e., tasks that return results without causing any side-effects. Both mechanisms are centralized and transparent to the applications.

Stop failures		without fault tolerance	using fault tolerance
the root node crashes		system aborts	system continues the normal operation
a non−root node crashes	less than half or half of non−root nodes crash within a certain time bound	deadlock	system continues the normal operation
	more than half of non−root nodes crash within a certain time bound		system aborts
	the root and all shadow root nodes crash within a certain time bound		system aborts
one or more connections between the root and non−root nodes break		deadlock	system continues the normal operation
one or more connections between non−root nodes break		deadlock	system continues the normal operation
one or more Maple processes fail		deadlock	deadlock

FIG. 7.1. *Assessment of Distributed Maple Fault Tolerance*

P-GRADE makes a checkpoint if and only if an external tool asks the server for the saving of a checkpoint. If the system fails during the computing phase, it can be restarted from the last checkpoint. If the system fails during the checkpointing phase, it cannot use any already saved information of this checkpoint and it can be restarted only from a previous checkpoint. The "Logging Results" mechanism saves every task descriptor and task result into separate files and it is able to recognize the corruption of a file; thus it may reuse each properly stored datum in a later session.

If a node fails (except for the server), the P-GRADE is able to recognize this and to continue execution automatically from the last checkpoint. In such a case, Distributed Maple is simply able to continue execution by the migration of the corresponding tasks (also if the root node fails). Distributed Maple regularly checks whether a failed node has rebooted again and in this case restarts the corresponding node process.

Summarizing, the most important advantage of the P-GRADE checkpointing mechanism is that it is made for general parallel applications. But its fault tolerance features are limited in that the main purpose was the dynamic migration of the processes among the node. The main disadvantage of the fault tolerant mechanisms in Distributed Maple that they are special mechanisms and they cannot be used for any parallel computing model. The advantages of our mechanisms are the following: the saving method is continuous; if the system fails in any time, the next session is able to use all properly saved intermediate task results; if any node or connection fails during the execution, the system is able to tolerate it and continue the normal operation without human intervention. Both systems are centralized, therefore the scalability of these systems is an open question.

7. Conclusion. We have implemented in Distributed Maple some fault tolerance mechanisms such that we can guarantee the following: the system does not deadlock and continues normal operation, if any node crashes or some connection between any two nodes breaks; if the system fails after all, we can restart it and continue the computation from a saved state. The improvement of the system is demonstrated by Figure 7.1.

The system can still fail if more than $n/2$ non-root nodes fail, or if the root and all shadow nodes fail within a certain time bound (i.e., if there is not enough time for reconnection).

The system can also fail if neither the root nor one of the shadow nodes has connected at least $n/2$ non-root nodes (this may happen if the network is split to more than two almost equal parts).

There remains only one kind of *Stop* failure situations which may let the system deadlock: if a kernel process fails. To solve this problem, we plan to introduce a new watching mechanism which scans the kernels and restarts them if necessary.

Our work shows how distributed computations that operate with an essentially functional parallel programming model can tolerate faults with relatively simple mechanisms, i.e. without global snapshots as required in message passing programs. The runtime overhead imposed by the logging mechanism is very moderate; adding tolerance of node failures (in case of both root node and non-root node) on top does then not require much extra overhead.

In the testing phase of our fault tolerance mechanisms, we executed some long-running computations, whose solutions are required approximately 3 or 4 days. During these execution, we triggered several *Stop* failure situations (both root and non-root failures) off. Our test experiences showed that the system is able to tolerate these kinds of failures and finish the computations.

One reason for this simplicity is the delegation of all logging activities to a single root node that also performs the task scheduling decisions; the model is therefore not scalable beyond a certain number of nodes. However, it is suitable for the system environments that we have used up to now ($\leq$ 30 nodes); many parallel computer algebra applications do not scale well enough to profit from considerably more nodes.

Acknowledgments. Sponsored by Upper Austrian Government (Ph. D. scholarship) and SFB/FWF project P1302.

REFERENCES

[1] David M. Arnow, *DP: A Library for Building Portable, Reliable Distributed Applications.* In *1995 USENIX Technical Conference*, pages 235–247, New Orleans, Louisiana, January 16–20, 1995. USENIX.

[2] Özalp Babaoglu, Lorenzo Alvisi, Alessandro Amoroso, Renzo Davoli, and Luigi Alberto Giachini, *Paralex: An Environment for Parallel Programming in Distributed Systems.* In *1992 International Conference on Supercomputing*, pages 187–187, Washington DC, July 19–24, 1992. ACM Press.

[3] David E. Bakken and Richard D. Schlichting, *Supporting Fault-Tolerant Parallel Programming in Linda.* IEEE Transactions on Parallel and Distributed Systems, 6(3):287–302, March 1995.

[4] Adam Beguelin, Erik Seligman, and Peter Stephan, *Application Level Fault Tolerance in Heterogeneous Networks of Workstations. Journal of Parallel and Distributed Computing*, 43(2):147–155, June 1997.

[5] Kenneth P. Birman, *Building Secure and Reliable Network Applications.* Manning, Greenwich, Connecticut, 1996

[6] Jeremy Casas, Dan Clark, Phil Galbiati, Ravi Konuru, Steve Otto, Robert Prouty, and Jonathan Walpole, *MIST: PVM with Transparent Migration and Checkpointing.* In *Third Annual PVM User's Group Meeting*, Pittsburgh, Pennsylvania, May 1995.

[7] A. Clematis and V. Gianuzzi, *CPVM — Extending PVM for Consistent Checkpointing.* In *4th Euromicro Workshop on Parallel and Distributed Processing (PDP'96)*, pages 67-76, Braga, Portugal, January 24–26, 1996. IEEE CS Press.

[8] G. E. Fagg and J. J. Dongarra, *FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World.* In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proceedings of the 7th European PVM/MPI Users' Group Meeting*, volume 1908 of *Lecture Notes in Computer Science*, pages 346–353, Balatonfüred, Hungary, September 10–13, 2000. Springer, Berlin.

[9] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek and Vaidy Sunderam, *PVM: Parallel Virtual Machine — A Users' Guide and Tutorial For Networked Parallel Computing. MIT Press, Cambridge, Massachusetts, 1994.*

[10] V. Gianuzzi and F. Merani, *Using PVM to Implement a Distributed Dependable Simulation System.* In *3rd Euromicro Workshop on Parallel and Distributed Processing (PDP'95)*, pages 529–535, San Remo, Italy, January 25–27, 1995. IEEE Computer Society Press.

[11] Hoon Homg, Andreas Neubacher, and Wolfgang Schreiner, *The Design of the SACLIB/PACLIB Kernels. Journal of Symbolic Computation*, 19:111–132, 1995.

[12] A. Iamnitchi and I. Foster, *A Problem Specific Fault Tolerance Mechanism for Asynchronous, Distributed Systems.* In *29th International Conference on Parallel Processing (ICPP)*, Toronto, Canada, August 21–24, 2000. Ohio State University.

[13] Pankaj Jalote, *Fault-Tolerance in Distributed Systems.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[14] R. Jagannathan and E. A. Ashcroft, *Fault Tolerance in Parallel Implementations of Functional Languages.* In 21st International Symposium on Fault-Tolerant Computing, pages 256–263, Montreal, Canada, June 1991. IEEE CS Press.

[15] József Kovács, Péter Kacsuk, *Server Based Migration of Parallel Applications.* In *Distributed and Parallel Systems—Cluster and Grid Computing, DAPSYS'2002, 4th Austrian Hungarian Workshop on Distributed and Parallel Systems*, pages 30–37, Linz, Austria, September 29—October 02, 2002. Kluwer Academic Publishers,Boston.

[16] Wolfgang Küchlin, *PARSAC-2: A Parallel SAC-2 based on Threads.* In *AAECC-8: 8th International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, volume 50 of *Lecture Notes in Computer Science*, pages 341–353, Tokyo, Japan, August 1990. Springer, Berlin.

[17] Juan Leon, Allan L. Fisher, and Peter Alfons Steenkiste, *Fail-safe PVM: a Portable Package for Distributed Programming with Transparent Recovery.* Technical Report CMU-CS-93-124, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, February 1993.

[18] M. Migliardi, V. Sunderam, A. Geist, and J. Dongarra, *Dynamic Reconfiguration and Virtual Machine Management in the Harness Metacomputing System.* In *Computing in Object-Oriented Parallel Environments — Second International Symposium (ISCOPE 98)*, volume 1505 of *Lecture Notes in Computer Science*, pages 127–134, Santa Fe, New Mexico, December 8–11, 1998. Springer.

[19] Nancy A. Lynch, *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc. San Francisco, California, 1996.

[20] *P-GRADE environment*: http://www.lpds.sztaki.hu/projects/pgrade.

[21] Taesoon Park and Heon Y. Yeom, *Application Controlled Checkpointing Coordination for Fault-Tolerant Distributed Computing Systems. Parallel Computing*, 24(4):467–482, March 2000.

[22] James S. Plank, Youngbae Kim, and Jack Dongarra, *Algorithm-Based Diskless Checkpointing for Fault Tolerant Matrix Operations.* In *25th International Symposium on Fault-Tolerant Computing*, Pasadena, California, June 1995. IEEE Computer Society Press.

[23] Samuel H. Russ, Jonathan Robinson, Brian K. Flachs, and Bjorn Heckel, *The Hector Distributed Run-Time Environment. IEEE Transactions on Parallel and Distributed Systems*, 9(11):1104–1112, November 1998.

[24] Fred B. Schneider, *Implementing Fault-Tolerant Services Using State Machine Approach. ACM Computing Surveys*, 22(4):299–319, December 1990.

[25] Wolfgang Schreiner, *A Para-Functional Programming Interface for a Parallel Computer Algebra Package. Journal of Symbolic Computation*, 21(4–6):593–614, 1996.

[26] Wolfgang Schreiner, *Distributed Maple — User and Reference Manual.* Technical Report 98–05, Research Institute for Symbolic Computation (RISC–Linz), Johannes Kepler University, Linz, Austria, May 1998. http://www.risc.uni-linz.ac.at/software/distmaple.

[27] Wolfgang Schreiner, *Analyzing the Performance of Distributed Maple.* Technical Report 00–32, Research Institute for Symbolic Computation (RISC–Linz), Johannes Kepler University, Linz, Austria, November 2000.

[28] Wolfgang Schreiner, Christian Mittermaier, and Franz Winkler, *Analyzing Algebraic Curves by Cluster Computing.* In *Distributed and Parallel Systems—From Instruction Parallelism to Cluster Computing, DAPSYS'2000, 3rd Austrian Hungarian Workshop on Distributed and Parallel Systems*, pages 175–184, Balatonfüred, Hungary, September 10–13, 2000. Kluwer Academic Publishers, Boston.

[29] Wolfgang Schreiner, Christian Mittermaier, and Franz Winkler, *On Solving a Problem in Algebraic Geometry by Cluster Computing.* In *Euro-Par 2000, 6th International Euro-Par Conference*, volume 1900 of *Lecture Notes in Computer Science*, pages 1196–1200, Munich, Germany, August 29—September 1, 2000. Springer, Berlin.

[30] Wolfgang Schreiner, Károly Bósa, Gábor Kusper, *Fault Tolerance for Cluster Computing on Functional Tasks.* Euro-Par 2001, 7th International Euro-Par Conference, Manchester, UK, August 28— August 31, 2001. Lecture Notes in Computer Science, Springer, Berlin, 5 pages, Springer-Verlag.

[31] Wolfgang Schreiner, Christian Mittermaier, Károly Bósa, *Distributed Maple: Parallel Computer Algebra in Networked Environments.* Journal of Symbolic Computation, volume 35, number 3, pp. 305-347, Academic Press, 2003.

[32] P. Stelling, C. Lee, I. Foster, G. von Laszewski, and C. Kesselman, *A Fault Detection Service for Wide Area Distributed Computations.* In *Seventh IEEE International Symposium on High Performance Distributed Computing*, July 28–31, Chicago, Illinois, 1998. IEEE Computer Society Press.

Edited by: Zsolt Nemeth, Dieter Kranzlmuller, Peter Kacsuk, Jens Volkert
Received: March 28, 2003
Accepted: June 5, 2003

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 71–81. http://www.scpe.org

ISSN 1895-1767
© 2005 SWPS

CONTEXT SENSING, AGGREGATION, REPRESENTATION AND EXPLOITATION IN WIRELESS NETWORKS

ALOIS FERSCHA, SIMON VOGL AND WOLFGANG BEER*

Abstract. As the use of wirelessly connected embedded mobile devices grows in vast quantity, their situative use and context sensitive behavior becomes an essential feature. For the implementation of context awareness in mobile appliances, the need for the efficient gathering, representation and delivery of context information evolves. This paper describes issues related to context sensing, representation and delivery, and proposes a new approach for context based mobile computing: Time and event triggered context sensing for mobile devices and an abstract (application and platform independent) representation of context information is introduced. The paper presents showcases of time and event triggered context sensing in wireless environments.

Key words. Mobile Computing, Context Awareness, Spatial Proximity Sensing, Event Trigger, Time Trigger, Identification, WLAN, RFID, RDF;

1. Introduction.

An encouraging development of mobile computing and communication devices enables a new vision for future ubiquitous computing environments. A great variety of different electronic devices will be embedded to our environment and to articles of daily use [1]. The intention is, to create intelligent self organizing environments, composed of a multitude of embedded systems. These systems should interact with people in a more natural—and thus more convenient—way than it is the situation today. In the past few years different research efforts have dealt with "smart spaces", environments, exploiting new hardware technologies, like submicron IC design, reliable WLAN communication, low power storage systems, new sensor and actuator technologies and smart materials [26]. With wireless communication technologies it is possible for embedded devices to communicate with the user and with each other, as well as to gather information about their local environment. The sensing of information about the local environment is important in that way, as it tells about the existing infrastructure surrounding a certain device. The sensing of information about e.g. the location (or geographical position) of a mobile device could minimize the infrastructure ultimately demanded to provide those services [26]. On the other hand this also means that it is not necessary to build upon a globally accessible network, but to e.g. use peer to peer communication where appropriate. Context computing [15] [19], i.e. the collection, transformation, interpretation, provision and delivery of context information [5][14][15] is the key to future development of smart environments and applications [4] [11]. Context information in a context aware application is captured via a collection of sensors [3], mapped to a representation of the real world ("world model" [19], and used to control the real world via a set of actuators. Recent research work related to "world modelling" has shown that a distinction among the abstract classes of person, thing and place is useful, when real world objects are mapped to objects in virtual environments [26][29][10]. This distinction is able to fulfill the needs, for abstract real world object base classes sufficiently [17].

In this work we present a generic context information representation framework for the person-thing-place world view, and develop context gathering mechanisms based on time and event triggered context sensors (Section 2). As an abstract context representation mechanism the Resource Description Framework (RDF) [31] is adopted in Section 3. We discuss the basic RDF definition as well as an appropriate RDF Schema (RDFS) as a means to provide a vocabulary and structures for expressing the context information gathered from different sensors. Ontologies [34] [13]do have to provide a special object model and a formal semantic, to support adequate modelling and reasoning of object context representations [33].

Wireless sensor networks are emerging as a potentially suitable setting to implement context based applications. We discuss issues of embedded networked environments as sensor actuator systems in Section 2. Particularly with this work we address the issues of automated context sensing and context information transformation, raised when nomadic users roam in a dynamically changing environment. In these cases of time varying object relation in the real world, mechanisms for an automated update of the world model are of critical importance. With the choice of RDF we also address the need to present context information in an application and platform independent way, and present effective methods to handle the context information updates, even in cyclic linked context information structures (Section 4). A demonstration scenario of our framework is developed in Section 5, conclusions are drawn in Section 6.

*Johannes Kepler University Linz Department for Practical Informatics, Altenbergerstrasse 69, 4040 Linz Austria. ferscha@soft.uni-linz.ac.at, vogl@soft.uni-linz.ac.at, beer@ssw.uni-linz.ac.at

2. Networks of Senors and Actuators. The vision of networked embedded systems, like e.g. an embedded Internet, in the future pervasive computing landscape is dominated by the ubiquity of a vast manifold of heterogeneous, small, embedded and mobile devices, the autonomy of their programmed behaviour, the dynamicity and context-awareness of services and applications they offer, the ad-hoc interoperability of services and the different modes of user interaction upon those services. This is mostly due to technological progress like the maturing of wireless networking, exciting new information processing possibilities induced by submicron IC designs, low power storage systems, smart material, and motor-, controller-, sensor- and actuator technologies, envisioning a future computing service scenario in which almost every object in our everyday environment will be equipped with embedded processors, wireless communication facilities and embedded software to percept, perform and control a multitude of tasks and functions [16]. Since many of these objects will be able to communicate and interact with the network and each other, the vision of "context-aware" network appliances and network spaces [28] [43] [21]—where dynamically configured systems of mobile entities by exploiting the available infrastructure and processing power of the environment—appears close to reality. We can hypothesize that the individual utility of such embedded network services will be greatly increased if they were personalized, i.e. user centered and dynamically adapted to user preference, location aware, i.e. multimodal and multifunctional with respect to the environment, and time dependent, i.e. if they were time dynamic and exhibited timely responsiveness [14] [15] [42]. One impacting trend hence will be the evolution of context aware environments [23] [35] [22]—often referred to as "smart appliances" or "smart spaces"—that intelligently monitor the objects of a real world (including people), and interact with them in a pro-active, autonomous, souvereign, responsible and user-authorized way. People will be empowered through an environment that is aware of their presence, sensitive, adaptive and responsive to their needs, habits and emotions, as well as ubiquitously accessible via natural interaction [20].

Recent developments in the technology of sensors and actuators, processing devices, embedded systems and wireless communication technologies already accelerate the maturing of embedded network applications [36]. "Context-aware" services [22] [17], i.e. applications able to take into account information about context, such as the location of the user or users [7] [18], the time of the day, the light, sound or humidity conditions, the available communication bandwidth, etc., and, most importantly, automatically adapt to the changing conditions in which they execute, are in the process of being cooperatively developed in the software engineering, embedded systems, communication and hardware integration research communities.

To build context aware applications, the adoption of a world model (or "virtual world") representing a set of objects and their state in a physical (or "real") world (or at least the subworld essential for the specific application) is the common approach suggested in the literature [14]. What makes an application context aware is the ability to interact with objects in the real world, requiring adequate models and representations of the objects of interest, and the possibility to sense, track, manipulate and trigger the real objects from within the world model. Several frameworks for such world models have appeared recently, the most prominent ones identifying persons, things and places as the primary abstract classes for real world objects [29][10]. People living in the real world, acting, perceiving and interacting with objects in their environment are represented in a "virtual world" by "virtual objects" or "proxies". Proxies of persons, things and places are linked to each other in the virtual world, such that this "linkage" is highly correlated with the "linkage" of physical persons, things and places in the real world. A context-aware application now monitors the state and activity of the real world objects via set of sensors, coordinates the proxies according to the rules embodied in the application, and notifies, triggers or modifies the physical world objects via a set of actuators.

The next generation of networked applications will evolve at the very edge of todays global networks (like e.g. the Internet) by wirelessly networked, embedded appliances, characterised by the possibilites of (*i*) ubiquitous access, (*ii*) context awareness, (*iii*) intelligence, (or "smartness") and (*iv*) natural interaction. Ubiquitous access here refers to a situation in which users are surrounded by a multitude of interconnected embedded systems, which are mostly invisible and weaved into the background of the surrounding, like furniture, clothing, rooms, etc., and all of them able to sense the setting and state of physical world objects via a multitude of sensors. Sensors, as the key enablers for implicit input from a "physical world" into a "virtual world", will be operated in a time-driven or event-driven way, and actuators, as the generic means for implicit output from the "virtual" to the "physical world", will respond to the surrounding in either a reactive or proactive fashion. The way how humans interact with the computing environment will thus fundamentally change towards implicit, ubiquitous access [42].

Context awareness refers to the ability of the system to recognise and localise objects as well as people

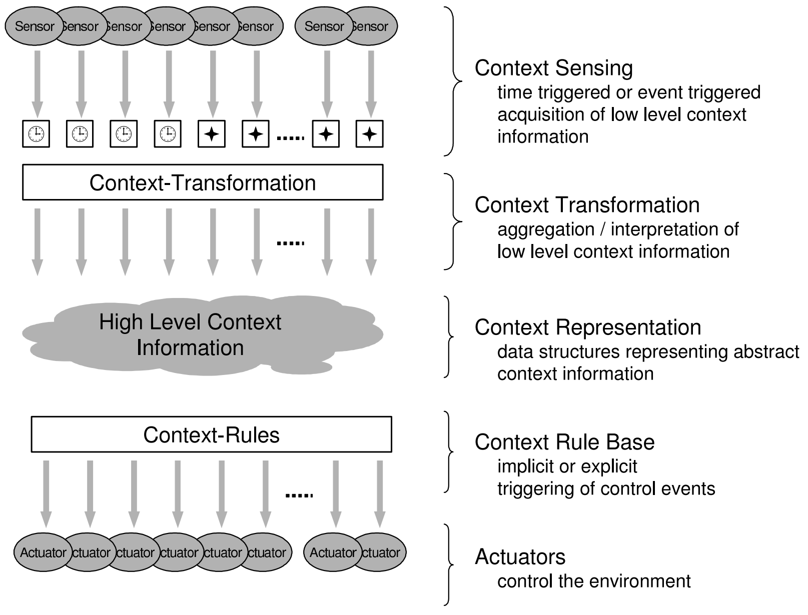

Sensor
Context-Transformation
High Level Context Information
Context-Rules
Actuator
Context Sensing
time triggered or event triggered acquisition of low level context information
Context Transformation
aggregation / interpretation of low level context information
Context Representation
data structures representing abstract context information
Context Rule Base
implicit or explicit triggering of control events
Actuators
control the environment

FIG. 2.1. *Context Information Life Cycle*

and their intentions. The context of an application is understood as "any information that can be used to characterize the situation of an entity", an entity being "a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [15]. A key architecture design principle for context-aware applications will be to decouple mechanism for collecting or sensing context information and its interpretation, from the provision and exploitation of this information to build and run context-aware applications [22]. To support building context-aware applications, software developers should not be concerned with how, when and where context information is sensed. Sensing context must happen in an application independent way, and context representation must be generic for all possible applications.

Intelligence refers to the fact that the digital surrounding is able to adapt itself to the people that live (or artefacts that reside) in it, learn from their behaviour, and possibly recognise as well as show emotion. Natural interaction finally refers to advanced modalities like natural speech- and gesture recognition, as well as speech-synthesis which will allow a much more human-like communication with the digital environment than is possible today.

3. Context Sensing in Smart Environments. Embedding processors into daily-use appliances lets users access an ever growing variety of information about the physical world [35], while at the same time a variety of "digital traces" can be collected upon monitoring the users activities. A software framework allowing for an ubiquitous collection of context information via sensors and the deployment of context information via actuators is the challenge we approach in the sequel. We develop a context information framework that addresses dynamic "physical world" changes by offering a flexible, extensible architecture based on two concepts: (*i*) the identification of (real world) objects, irrespective of the sensing technology, and (*ii*) the distinction of two types of context information sources: continuous context information streams and occasional context events.

The foundation for our context sensing network is the ability to digitally identify objects via various identification and addressing methods. Each active artifact (objects playing an active role with respect to our framework) needs its own unique ID, that makes it identifiable and allows further tracking and logging of activities.

Integrating different technologies implies that we have to cope with varying addressing and identification schemes. We deal with this issue by assigning name spaces to the families involved and format these identifiers like URIs, with a leading name-space tag. As an example, consider a mobile device known to the framework as ip:140.78.95.11, i.e. a PDA that is linked wirelessly with our campus network and recognized via its name-space tag ip:, or one of our rooms identifies as rfid:0800B9F1AFB1—here a unique RFID tag number—recognized via the name-space tag rfid:. This way, it is easy to integrate new sensors into our framework, covering barcodes (ean:90018211212), ISBN-numbers, or any other type of fingerprint.

As far as input data is concerned, we distinguish among two fundamentally different types of information, that need to be treated in different ways: (*i*) events potentially occurring at certain occasions in the real world and (*ii*) continuously occurring events describing context data streams. Consequently, two different context sensing mechanisms are needed: (*i*) watchdog mechanism monitoring the environment for the occurrence of events and their immediate notification to the framework, and (*ii*) the probing of real world state information continuously over time and the filtering of these streams with respect to data volume and importance to the framework. We refer to the latter as continuous (context) sensing, to the former as event based (context) sensing.

3.1. Continuous Sensing. Continuous data sources provide information about the current state of the real world, e.g.: indicators like temperature, light conditions, link quality, etc. This type of data is typically sampled at fixed time intervals. Alternatively, it may be sampled upon request. In a web-based environment, this data can be provided by a simple HTTP server script. This is an easy way to retrieve the current value. Persistent connections can be used when values should be read at fixed intervals, to minimize connection overhead (HTTP), a method commonly in use for WebCams, status monitors and the like. One of our continuous data sources provides a list of WLAN devices, their MAC and IP numbers, in combination with the name of the WLAN-access point they are associated with. This provides a compact overview of active devices and their location based on access point influence radii.

3.2. Event Based Sensing. The other data source we use does not deal with system states, but rather with their changes. Events occur whenever some change is detected — e.g.: in the real world by analyzing sensor data, or in a software component executing the world model framework. Due to their dynamic nature, these events cannot be read conveniently over standard HTTP mechanisms at the point of their occurrence. Instead, events that occur have to be actively propagated to the interested parties by calling event handler scripts on their web servers. Like in simulation environments, we can generate events that are derived from continuous data sources by defining specific triggers, like threshold values, or query strings that match different situations we are interested in.

In our test setting we use RFID readers that are connected to mobile devices as event-generating data sources. Whenever a transponder enters the spatial proximity of the electro-magnetic field of the RFID-readers, its unique ID is read and posted to an event listener service, on the server machine executing the context framework which we call `UBIC`.

The `UBIC` framework is exclusively based on standard internet technologies like IP-addressing and HTTP. In its current state of implementation `UBIC` relies on fixed server addresses and thus on the availability of an access network, which is, in our case, a campus WLAN. Together with the concept of an (RFID-)tagged environment, `UBIC` can seamlessly combine identification as well as positioning and tracking of the real world.

Figure 1 shows how a mobile device reads a tag that is associated with a place (e.g. an office-room) and transmits the tag ID to the `UBIC` framework. This triggers the updater process that checks the relations defined for the tagged item as well as the device that caused the event, updates all concerned data and sends this information to the persistent history as well as a notification process. That informs all interested parties that a room change has occurred.

4. Abstract Representation of Context Information. To proliferate context information in a timely due manner and general format — irrespective of its purpose of use or application — a representation of context information based on the resource description framework (RDF) is proposed, modeling the artifacts person, thing and place as RDF resources. In combination with RDF Schema (RDFS), RDF provides a powerful syntax for semantic knowledge modeling [31]. RDF and RDFS are based on the XML Web data exchange format. The basic RDF model consists of three object types, which form subject, predicate and object triples, called RDF Statements. These three object types are:

1. Resources: Every object described in RDF is called a resource, that could be a simple HTML page, data like pictures and graphics on the web or, like in this work, real objects which are not directly embedded into the internet. In this work many objects of the real world, distinguished into person, thing and place present their context information written in RDF.

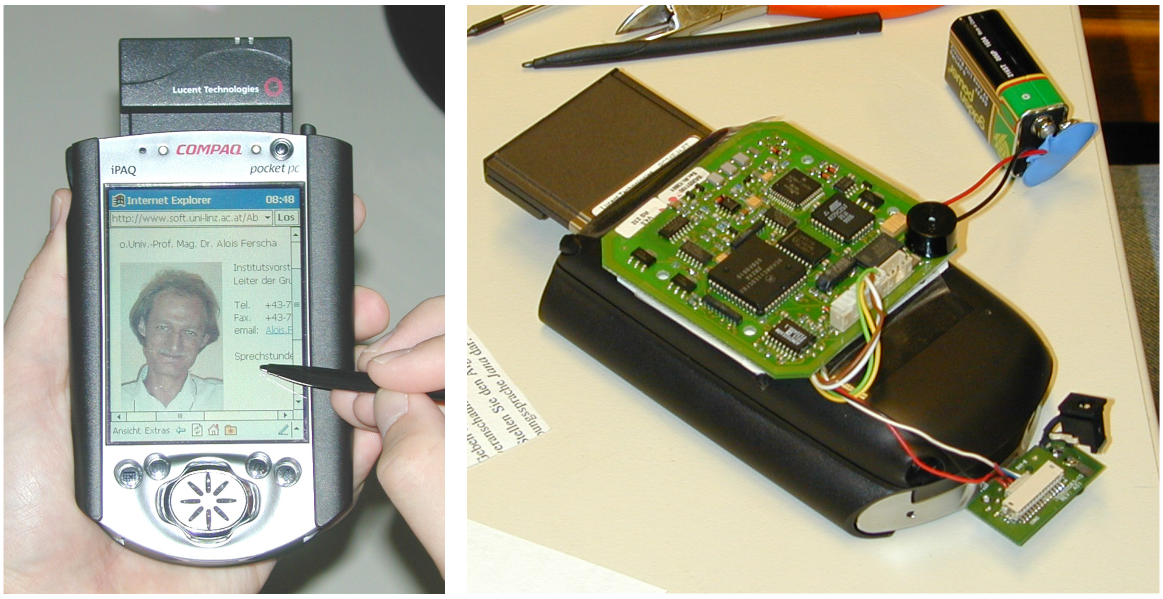


Fig. 3.1. *Combining of WLAN and RFID Capabilities in a Commercial PDA*

2. Properties: Properties are specific attributes which describe a resource and its relation to different resources. Every property has its own characteristics, like values which are permitted and values that are not permitted. The RDF basic syntax and model specification does not address this specific property characteristics. For this purpose the RDF Schema specification is needed.
3. Statements: The subject (resource), predicate (property) and the property value (object) triple, build an RDF-Statement. The property value could be a literal, another resource or an URI to another resource.

RDF has a simple, but powerful, model and syntax definition and is therefore a good choice for representing and delivering context sensing information. Furthermore, it is simple to import or export RDF statements (subject, predicate and object triples) from any kind of database. Another important aspect concerning the use of RDF and RDFS as a standard context information representation and delivery format is that a promising research effort (like the "Dublin Core" [10], "On to Knowledge" [34] and "Semantic Web") is under way to establish a standard vocabulary for semantic data description. `UBIC` can thus refer to standard for properties like "email", "name", "date", or "creator", etc. In `UBIC`, resource properties are used to describe context information. Basic sets of context information properties are used for the representation of location, containment, ownership and person to thing relations, together with a history of these context information properties. The following list of basic context information properties is used in our context sensing framework, distinguishing basically three types of objects:

- Place: Places hold information tags about the location (location) and references to a set of objects which are actually inside the place (contains). In order to track changes in the contains list it is necessary to store any action in a history tag (contained). Entries of the contained property list have two time stamps, called startTime and endTime, specifying the period of time when the entry was part of the contains property list.
- Person: A person holds information tags about its actual location inside the scenario (isIn) and a list of objects which are owned by the person (ownerOf). Additionally the person holds a list of locations where it was before (wasIn).
- Thing: A thing holds information on whether it is able to contain other objects (canContain), as for example a bag or a bagpack is able to. In the case that it can contain other objects, it also holds a contains information tag and a contained tag to track the objects. Furthermore a thing has an information tag about its owner (owner).

We can summarize that in `UBIC` an application specific abstraction of the real world is generated from three generic classes for persons, things, and places. The reification of physical world objects and their relation among

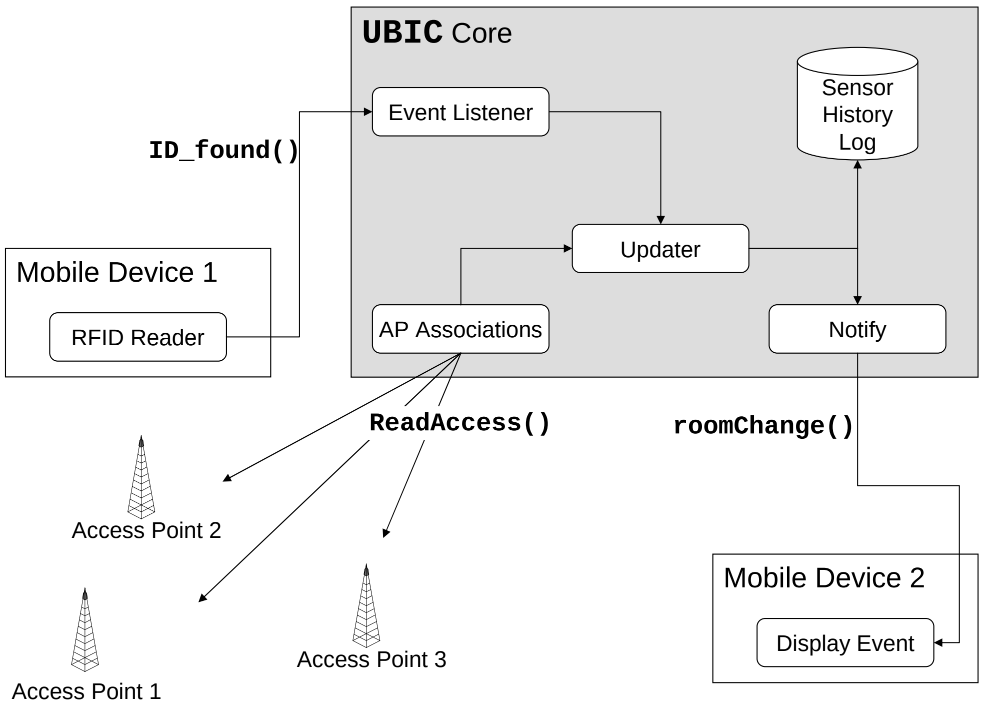


FIG. 3.2. *System Overview*

each other is expressed in RDF. The person object represents the concepts about a person that are necessary to link their physical properties and activities to the virtual world. The thing object encapsulates the basic abstraction for objects relevant to the application and has instances like shirt, bag, etc. The place object is used to represent the essential features of a physical location like an office room or a waiting lounge. Places, things and persons may be related in a manifold of ways.

5. Application Scenarios.

5.1. Context-Aware Suitcase. With our first application scenario we aim at the construction of a context-aware suitcase [17], as shown in Figure 5.2. The primary intent is to make an ordinary suitcase aware of the presence of things it contains, that it has (ever) contained, the location where it resides at the moment (and all the different locations its has ever been to), etc., and to meaningfully use this context information to provide services.

In a first step, we model the context of the suitcase in terms of RDF based abstract representation as described above. The physical appearance of the suitcase is expressed as an instance of the abstract object class `place`, the owner of the suitcase as an instance oft he abstract class `person`, and things to be carried with the suitcase as instances of the abstract class `thing`. The contextual interrelatedness among those object instances is described by a set of (bilateral) object relations expressed in RDF. The whole scenario is expressed in terms of RDF statements, i.e. denoted as subject, predicate and object triples. A subject is represented by a resource, that could be a URL, a simple HTML file, a URI reference etc., a predicate is represented by a property describing a resource or the relation among resources, and an object is represented by a property value. As an example, the statement that the person `ferscha` owns the thing `suitcase` is expressed in RDF as in Figure 5.1, which sketches quite a few relations: The `owner` relation expresses ownership of a real world object by another, the `contains` and `is_in` relations expresses geometrical containment of an object within another one, the `contained` and `was_in` relations trace the history of containment into RDF bags, the `containable` attribute defines whether an object may contain another one, the `controllable` attribute allows to prevent or enable the modification of an object RDF by the object itself, etc. The unique ID associated with every real

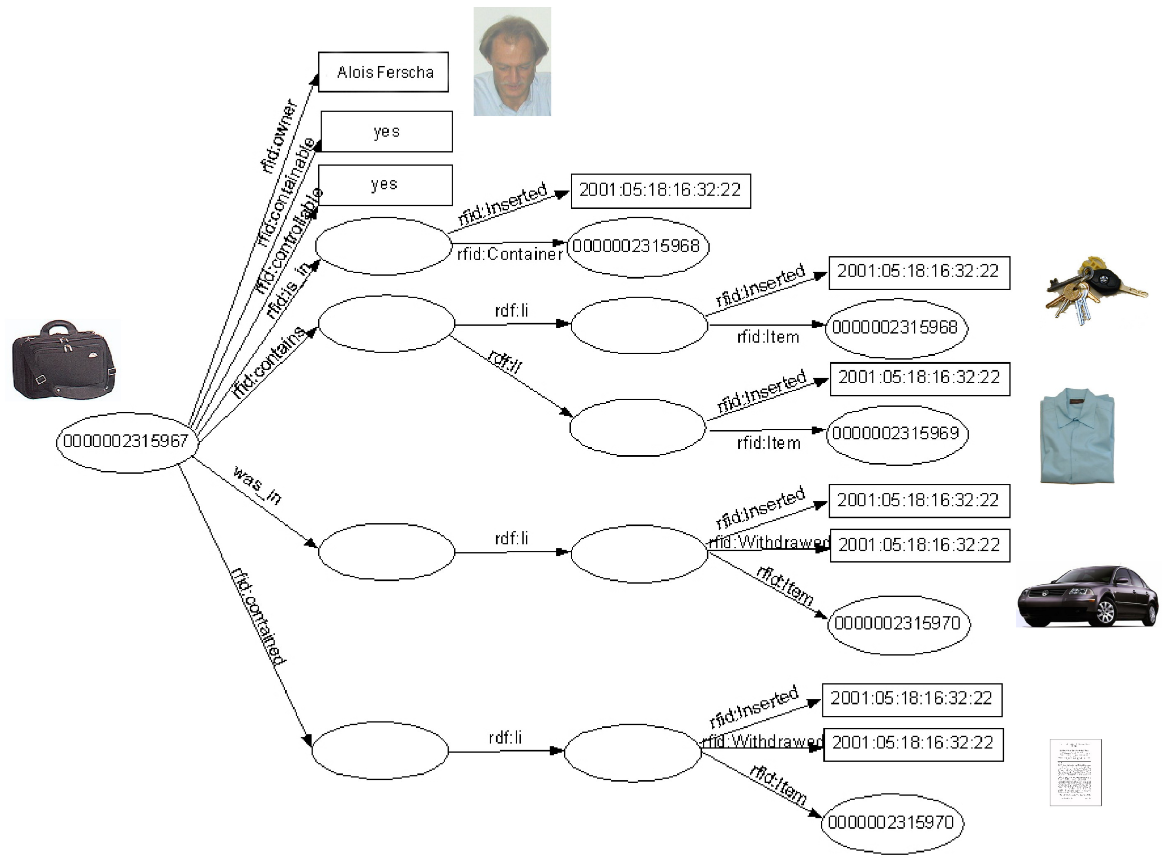


FIG. 5.1. *An RDF Statement Describing an Ownership Relation*

world object is the ID encoded in its RFID tag. It is supposed to be sensed by an appropriate sensor which triggers a script to update the involved object RDFs (Inserting e.g. the shirt into the suitcase would cause an RFID reader to identify the shirt tag and automatically update (among others) both the shirts RDF relation `is_in`, as well as the suitcases RDF relation `contains` by cross-referring URIs.)

Object instances can be created and interrelated at run-time, relations can be updated according to dynamic changes of context (like time, location, etc.) via time-triggered or event-triggered sensors. Upon change of context, like the insertion of a thing into the suitcase, or the movement of the suitcase to a different place, inheritance mechanism resolve transitivity of object relations like the `is_in` relation for objects `contained` in a (geographically) moving (physical world) object.

For building the physical demonstration prototype, an embedded single board computer has been integrated into an off-the-shelf suitcase (see Figure 5.1), executing a standard HTTP services on top of a TCP/IP stack over an integrated IEEE802.11b WLAN adaptor. A miniaturized RFID reader is connected to the serial port of the server machine, an RFID antenna is integrated in the frame of the suitcase so as to enable the server to sense RFID tags contained in the suitcase. A vast of 125KHz magnetic coupled transponders are used to tag real world objects (like shirts, keys, PDAs or even printed paper) to be potentially carried (and sensed) by the suitcase. The suitcase itself is tagged and possibly sensed by readers integrated into home furniture, car or airplane trunks, conveyor belts etc. so as to allow for an identification and localization at any meaningful point in space of the application.

5.2. Context-Aware Office. Our second scenario covers a more complex setup: a campus WLAN is used for a rough localization of persons roaming the university by tracking the access points their mobile devices are associated with. We superimpose fine-grained position information derived from tagged environments. Offices have been covered with myriads of RFID tags, giving us the possibility to track devices in 10cm ranges [18]. So it is possible to map any sensed tag identification number to a specified location. Figure 5.3 shows how RFID tags can specify locations and map this locations to a virtual environment.

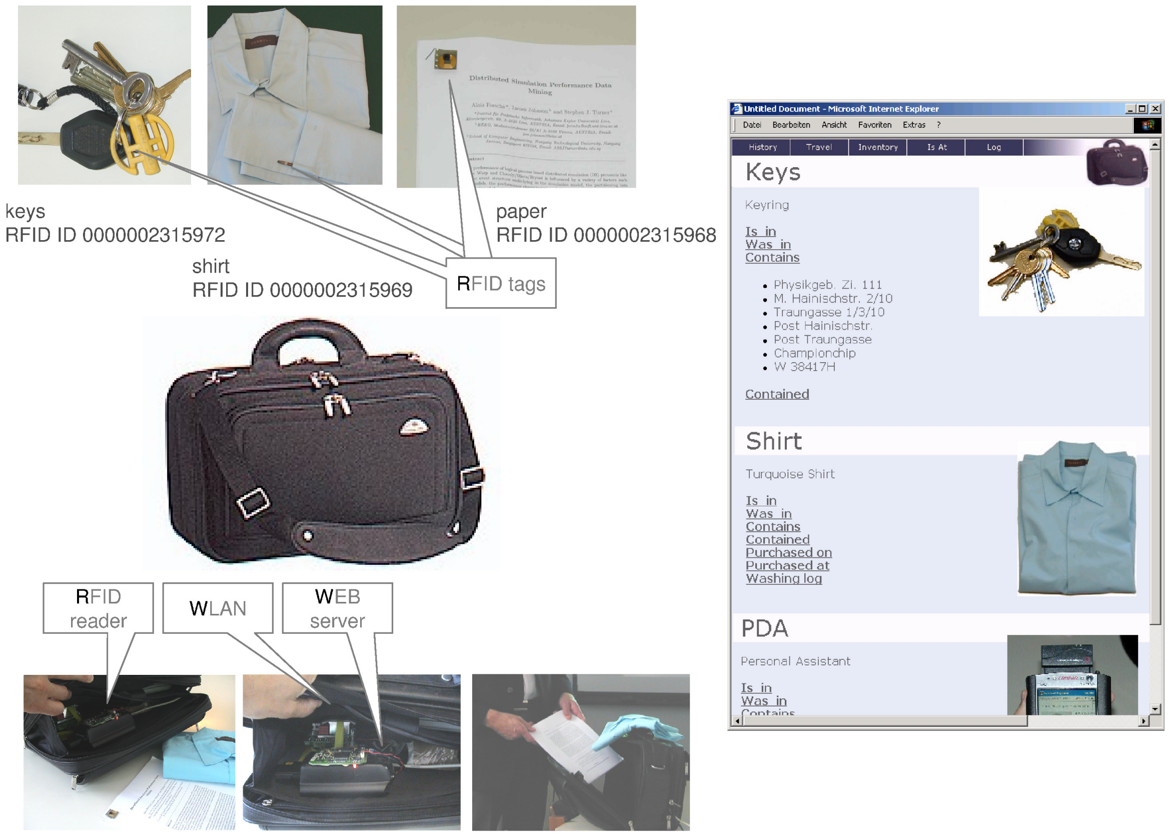


Fig. 5.2. *Context-Aware Suitcase as an Application Instance of the UBIC Framework*

In this application of the `UBIC` framework context sensing is reached by tracking the WLAN access points and reading their association tables. This scenario intends to combine event and time triggered context sensing and to integrate the context sensing information into a wireless location awareness framework. For exact spatial proximity information the RFID technology was used and in order to get location spheres context information the access point association tables were used.

6. Conclusion. Besides the demand for new middleware paradigms and concepts that encompass software issues, devices, users, interaction, and dynamic changes of context, new requirements for the automated configuration and operation of embedded networked applications have emerged. Central to those is the notion of "context", preliminarily defined as any information that can be used to characterize the state or situation of an entity and the spectrum of its behaviors. Software architectures and frameworks for context-aware applications [19] thus must be concerned with (*i*) abstract representations of context, (*ii*) sensing, collecting, interpreting and transforming of context information and (*iii*) disseminating and making appropriate use of context information to steer the application. The methodological approach used in this paper was to employ standardized Web-metadata modelling (like SGML, XML, RDF) to integrate an arbitrary set of sensors (electrical, chemical, magnetic, optical, acoustic etc.) with an arbitrary set of actuators (like data processors, controllers, motors, filters, etc.) within a single framework. We have presented the distinction among event based and time triggered context sensing, and have shown how these two sensing mechanisms work together in practical applications. Additionally this work has demonstrated how to represent context sensing information with the XML based RDF and RDFS semantic data description standard.

Given today's global networks as an already existing backbone communication infrastructure, ubiquitous access to this infrastructure still demands technological solutions for the "spontaneous" discovery and configuration of devices and services, the selection of meaningful services offered by autonomous software components, the automatic adaptation to mobile and sporadic availability, the interoperability across manufacturers, platforms and services and the scalability with respect to the number of involved activities and entities. On top of

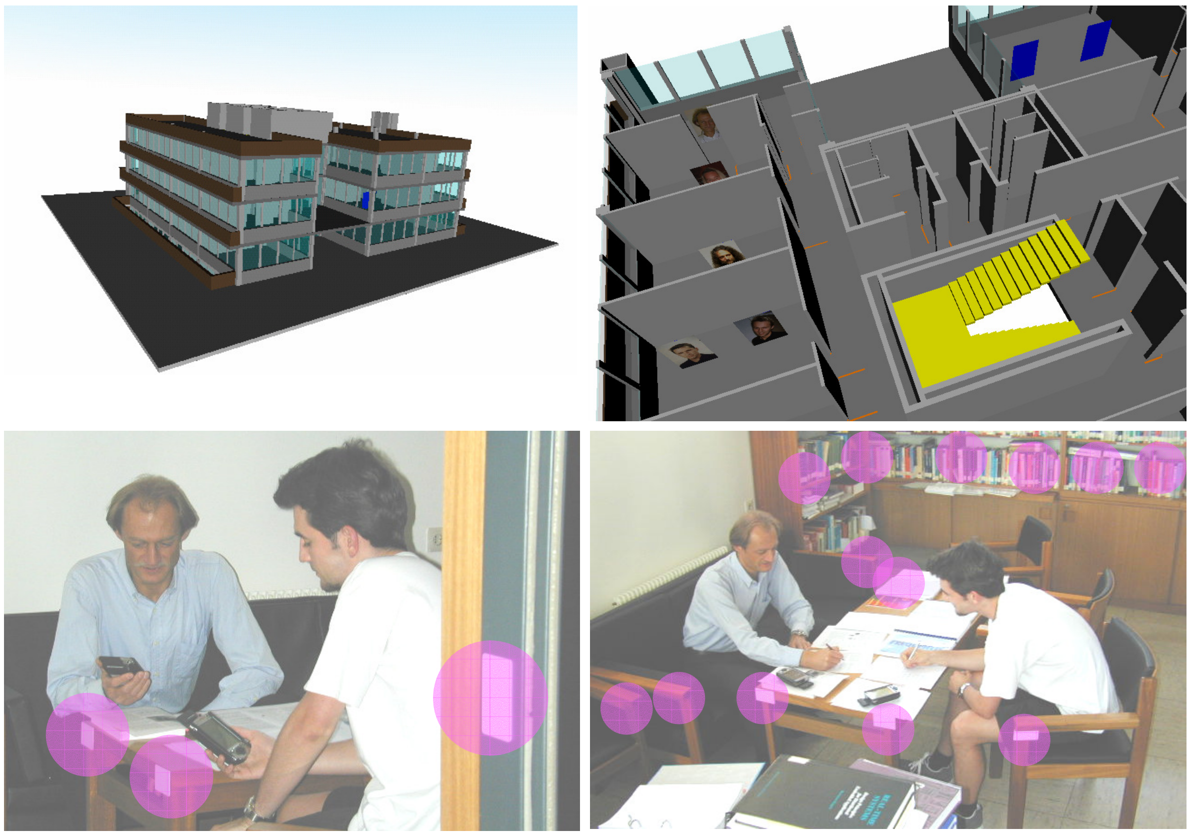


Fig. 5.3. *Mapping of RFID Tags to Locations in a Wireless Campus Office*

ubiquitous access, context-awareness and issues of intelligence (learning, memorizing, planning, forgetting) and knowledge processing are essential for the provision next generation context-aware applications. To this end—as a first approach for future `UBIC` applications—we are trying to exploit patterns in the context history of context aware, `UBIC` appliances and spaces, so as to be able to serve even anticipated future contexts. As opposed to "re-active" context awareness in the present implementation of `UBIC`, future versions will comprise a context prediction engine (based on statistical inferencing from the context history) and thus implement "pro-active" context awareness.

REFERENCES

[1] D. Arnold et.al., *Discourse with disposable Computers: How and why you will talk to your tomatoes*, USENIX Workshop on Embedded Systems, 1999.

[2] P. Bahl and V. N. Padmanabhan, *RADAR: An in-building RF-based user location and tracking system*, in Proceedings of IEEE INFOCOM 2000, pp. 775784, Mar. 2000.

[3] V. Bellotti, M. Back, W. Edwards, R. Grinter, A. Henderson, and C. Lopes , *Making Sense of Sensing Systems: Five Questions for Designers and Researchers*, Proc. CHI 2002, CHI Letters, 1 (1), 415-422, Minneapolis, USA, 20-25 April, 2002, ACM.

[4] St. Benford, H. Schnadelbach, B. Koleva, B. Gaver, A. Schmidt, A. Boucher, A. Steed, R. Anastasi Ch. Greenhalgh, T. Rodden, and H. Gellersen, *Sensible, sensable and desirable: a framework for designing physical interfaces*, Technical Report Equator-03-003, Equator, February 2003.

[5] P. J. Brown, J. D. Bovey, X. Chen, *Context-Aware Applications, From the Laboratory to the Marketplace*, IEEE Personal Communications, 1997 Vol. 4, Nr. 5.

[6] N. Bulusu, D. Estrin, L. Girod, and J. Heidemann , *Scalable coordination for wireless sensor networks: Self-configuring localization systems*, in Proceedings of the Sixth International Symposium on Communication Theory and Applications (ISCTA '01), July 2001.

[7] N. Bulusu, J. Heidemann, and D. Estrin, *GPS-less low-cost outdoor localization for very small devices*, IEEE Personal Communication, vol. 7, pp. 2834, October 2000.

[8] M. Brunato and K. K. Csaba, *Transparent location fingerprinting for wireless services*, in Proceedings of Med-Hoc-Net 2002, 2002.

[9] S. Card, J. Mackinlay and G. Robertson, *A Morphological Analysis of the Design Space of Input Devices*, ACM Transactions on Information Systems 9 pp. 99-122 (1991).

[10] D. Caswell, P. Debaty, *Creating a Web Representation for Places*, Proc. of the Ninth International World Wide Web Conference, 2000.

[11] B. Clarkson, K. Mase, and A. Pentland *Recognizing user's context via wearable sensors*, Proc. of the IEEE International Symposium on Wearable Computing (ISWC'00), pp. 69-76, 2000.

[12] A. Dix, T. Rodden, N. Davies, J. Trevor, A. Friday, and K. Palfreyman , *Exploiting Space and Location as a Design Framework for Interactive Mobile Systems*, ACM Transactions on Computer-Human Interaction (TOCHI) 7(3), September 2000, pp. 285-321.

[13] Dublin core initiative, `http://dublincore.org`

[14] A. K. Dey, G. D. Abowd, *Toward a Better Understanding of Context and Context-Awareness*, GIT, GVU Technical Report GIT-GVU-99-22, June 1999.

[15] A. K. Dey, *Understanding and Using Context. Personal and Ubiquitous Computing*, Special Issue on Situated Interaction and Ubiquitous Computing, 5(1), 2001.

[16] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, *Instrumenting the world with wireless sensor networks*, in Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, vol. 4, (Salt Lake City, UT), pp. 20332036, May 2001.

[17] A. Ferscha, *Towards a Context-Aware Embedded Internet*, Proceedings of Informationstagung Mikroelektronik 2001, pp.445-456, ÖVE Schriftenreihe Nr. 26, Wien, 2001.

[18] A. Ferscha, W. Beer, W. Narzt, *Location Awareness in Community Wireless LANs*, Informatik 2001, Tagungsband der GI/OCG-Jahrestagung, Vienna, Austria, September 2001, pp.190-195, Österreichische Computergesellschaft, Wien 2001.

[19] A. Ferscha, *Contextware: Bridging Physical and Virtual Worlds*, Reliable Software Technologies - AE2002, Lecture Notes in Computer Science 2361, Springer Verlag, Berlin, pages: 51-64, 2002.

[20] W. Gaver, J. Beaver and S. Benford , *Ambiguity as a resource for design* , in Proceedings of CHI 03, pp. 233-240, Fort Lauderdale, Florida, April 2003.

[21] H.W. Gellersen, M. Beigl and H. Krull, *The Mediacup: Awareness Technology Embedded in an Everyday Object*, Proc. of First International Symposium on Handheld and Ubiquitous Computing (HUC99), Karlsruhe, Germany, September 1999, LNCS 1707, Springer-Verlag, pp. 308-310.

[22] H.W. Gellersen, M. Beigl and A. Schmidt, *Sensor-based Context-Awareness for Situated Computing*, Proc. of Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland, June 2000.

[23] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster , *The anatomy of a context-aware application*, in Proceedings of MOBICOM 1999, pp. 5968, Aug. 1999.

[24] M. Hassan-Ali, K. Pahlavan, *A New Statistical Model for Site-Specific Indoor Radio Propagation Prediction Based on Geometric Optics and Geometric Probability*, IEEE Transactoins on Wireless Communications, vol. 1, no. 1, pp. 112-124, Jan. 2002

[25] W. B. Heinzelman, A. Chandrakasan, and H. Balakrishnan , *An Application-Speci.c Protocol Architecture for Wireless Microsensor Networks* , IEEE Transactions on Wireless Communications, vol. 1, pp. 660-670, 2002.

[26] Hewlett Packard, *CoolTown Appliance Computing: White Papers*, `http://cooltown.hp.com/papers.htm`

[27] J. Hightower, G. Borriello, and R. Want, *SpotON: An Indoor 3D Location Sensing Technology Based on RF Signal Strength*, University of Washington, Technical Report: UW-CSE 2000-02-02, Feb. 2000.

[28] L.E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl and H.W. Gellersen, *Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts*, Proc. of UBICOMP 2001, Atlanta, GA, USA, September 2001.

[29] T. Kindberg, J. Barton, J. Morgan, G. Becker, I. Bedner, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, M. Spasojevic, *People, Places, Things: Web Presence for the Real World*, in Proceedings of the WWW'2000, 2000.

[30] S. Kumar, D. Shepherd, F. Zhao, *Collaborative signal and information processing in microsensor networks*, IEEE Signal Processing Magazine, vol. 19, March 2002.

[31] O. Lassila, and R. R. Swick, *Resource Description Framework (RDF): Model and Syntax Specification*, Recommendation, World Wide Web Consortium, Feb. 1999.

[32] S. Lindsey and C. S. Raghavendra, *PEGASIS: Power-Efficient Gathering in Sensor Information Systems*, Proc. IEEE Aerospace Conference, vol. 3, pp. 1125 -1130, 2002.

[33] A. Omicini, F. Zambonelli, M. Klusch, R. Tolksdorf (Eds.), *Coordination of Internet Agents. Models, Technologies, and Applications*, Springer Verlag, Berlin 2001.

[34] OntoKnowledge, *On To Knowledge Organisation*, `http://www.ontoknowldge.org`

[35] J. Pascoe, N. Ryan, D. Morse, *Issues in developing context-aware computing*, In H-W.Gellersen, editor, Handheld and Ubiquitous Computing, Nr. 1707 in Lecture Notes in Computer Science, Heidelberg, Germany, September 1999. Springer-Verlag

[36] G. Pottie and W. Kaiser, *Wireless integrated network sensors*, Communications of the ACM, vol. 43, pp. 5158, May 2000.

[37] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, *The cricket location-support system*, in MOBICOM 2000, pp. 3243, Aug. 2000.

[38] H. Regenbrecht and M. Wagner, *Interaction in a Collaborative Augmented Reality Environment*, CHIi 2002 Extended Abstracts, 504-505, 2002.

[39] T. Roos, P. Myllymaki, H. Tirri, P. Misikangas, and J. Sievanen, *A probabilistic approach to wlan user location estimation*, International Journal of Wireless Information Networks, vol. 9, July 2002.

[40] C. Savarese, J. Rabaey, and J. Beutel, *Locationing in distributed ad-hoc wireless sensor networks*, in Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, vol. 4, (Salt Lake City, UT), pp. 20372040, May 2001.

[41] A. Savvides, C. C. Han, and M. B. Srivastava, *Dynamic fine-grained localization in ad-hoc wireless sensor networks*, in Proceedings of the International Conference on Mobile Computing and Networking (MobiCom) 2001, (Rome, Italy), July 2001.

[42] A. Schmidt, *Implicit Human-Computer Interaction through Context*, Personal Technologies 4(2 and 3), June 2000, pp. 191-199.

[43] A. Schmidt, and K. Van Laerhoven, *How to Build Smart Appliances?*, IEEE Personal Communications 8(4), August 2001.

[44] R. Want, A. Hopper, V. Falcao, and J. Gibbons, *The active badge location system*, ACM Transaction on Information Systems, vol. 10, pp. 91102, Jan. 1992.

[45] Y. Zou, K. Chakrabarty, *Energy-Aware Target Localization in Wireless Sensor Networks*, in Proc. IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), pp. 60-70, March, 2003.

Edited by: Zsolt Nemeth, Dieter Kranzlmuller, Peter Kacsuk, Jens Volkert
Received: April 14, 2003
Accepted: June 5, 2003

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 83–92. http://www.scpe.org

ISSN 1895-1767
© 2005 SWPS

A CORBA-BASED MIDDLEWARE FOR AN ADAPTIVE STREAMING SERVER

BALÁZS GOLDSCHMIDT[†] , ROLAND TUSCH[‡] , AND LÁSZLÓ BÖSZÖRMÉNYI[‡]

Abstract. A CORBA-based infrastructure for an adaptive multimedia server is introduced, enabling dynamic migration or replication of certain multimedia applications among a set of available server nodes. The requirements from both the server's and the middleware's point of view are discussed. A specification of a CORBA interface and a corresponding implementation is presented. Finally, as a proof of concept, measurements of the implementation are shown and discussed. The measurements show the importance of the related servers' distribution on the network.

Key words. Multimedia, CORBA, proactive adaptation, mobile agents

1. Introduction. In [15, 16] we presented a distributed multimedia streaming server architecture, which builds upon the concepts for distributed multimedia servers described in [7]. The key extension of the presented architecture therein is the capability that server components (also referred to as applications) may be dynamically migrated or replicated to other server nodes on demand. Program and data dependencies are also considered. The proposed adaptive multimedia server architecture (ADMS) mainly consists of four distinguished types of components: one *cluster manager*, a set of *data managers*, *data collectors*, and *data distributors*. The cluster manager controls the layout of the ADMS cluster concerning the locations of instances of the three other component types (see figure 2.3). Data managers provide facilities for efficient storage and retrieval of units of media data. A data distributor distributes a media stream received from a production client to a given set of data managers. Finally, data collectors perform inverse operations to distributors. A collector retrieves stripe units from data managers, reassembles and streams them to the requesting client(s). Thereby it can serve as a proxy which caches media streams possibly in different qualities, in order to satisfy a number of client requests with varying QoS demands.

Since the proposed architecture employs the full server utility model [14], each server node can run at most one component at a time. Following this model and focusing on especially the data collector component, there are two main questions regarding an optimal provision of quality of service to the requesting clients: (1) Does the component run on optimal locations? (2) If not, what locations should be recommended to run the component, and what are the costs for performing a component migration or replication? These questions typically can not be answered by the server nodes themselves, since for an answer both local and global views on the performance behaviour of the distributed server architecture are needed. Thus, a kind of infrastructure is required which monitors the resource usage on each server node, as well as the quality of the connection links between the nodes.

Infrastructures for managing adaptive multimedia servers are rather spare. Adaptive servers usually support code migration or replication, however, they hardly provide QoS support. An example for such infrastructures is Symphony [3], which provides a number of services for executing virtual servers in internet settings. Symphony does not support a QoS-constrained replication/migration of a multimedia service. The same is valid for Jini[17], a middleware for maintaining a dynamic network of resources and users. Jini's object replication algorithms rely on a network of reasonable speed and latency. QoS-aware middleware frameworks enabling QoS-driven adaptations, as presented in [8], focus on resource management dynamics, rather than on application-level dynamics.

In [1] it is illustrated that mobile agents are very well suited for network management issues like fault, configuration and performance management. Monitoring application demands, network and server loads are predestinated tasks for a mobile agent-based middleware. These lead us to the idea of using such a middleware for an adaptive multimedia server. Since the process of replication/migration is not intended to happen in real-time, an extended version of the CORBA-based mobile agent system Vagabond [4] is used. Vagabond is a 100% pure Java mobile agent system, first developed to be used in distributed multimedia database systems. However, its modular design allows to modify and use it as a middleware for an adaptive multimedia server, resulting in its successor *Vagabond2*.

[1]Budapest University of Technology and Economics, Department of Control Engineering and Information Technology; balage@inf.bme.hu
[2]Klagenfurt University, Institute of Information Technology; {roland, laszlo}@itec.uni-klu.ac.at

This paper addresses the requirements of the adaptive distributed multimedia streaming server ADMS to the underlying infrastructure Vagabond2, as well as the needs of Vagabond2 to ADMS. The remainder of this paper is organized as follows. Section 2 discusses the requirements of ADMS to the underlying infrastructure. In section 3 the needs of Vagabond2 to ADMS are disputed. Section 4 presents the CORBA interface specification between ADMS and Vagabond2. Section 5 deals with the internal architecture and implementation of Vagabond2. In section 6 the results of first measurements in the ADMS testbed are presented. Section 7 concludes this paper including a plan on future work.

2. Requirements to the Underlying Infrastructure. In [16], the limitations of a static distributed multimedia streaming server (SDMS) are discussed. Consider figure 2.1, which illustrates a retrieval scenario of an MPEG-4[9] presentation (Sample.mp4) from a SDMS. The presentation consists of only one video elementary stream, whose data is striped among two data manager instances *dm1* and *dm2*. The meta data of all available presentations on the SDMS is stored in an MPEG-7 meta data database, and can be queried by a so-called *meta data manager* (mdm). The *mdm* is not a separated server component, but simply an object which is typically integrated into the cluster manager component. Further, there exists one instance of a cluster manager (*cm*), data collector (*dc1*), and adaptation engine (*ae*). The adaptation engine tries to find an optimum data collector for a given client request, taking into account the locations of the data managers, among which the data of the requested media stream is striped. Similar to the *mdm*, the *ae* needs not to be a separated server component, and can also be an integral object of the *cm*. However, for the reason of executing computation-intensive optimization algorithms, it might be designed as a separated component, which can also be migrated or replicated on demand.

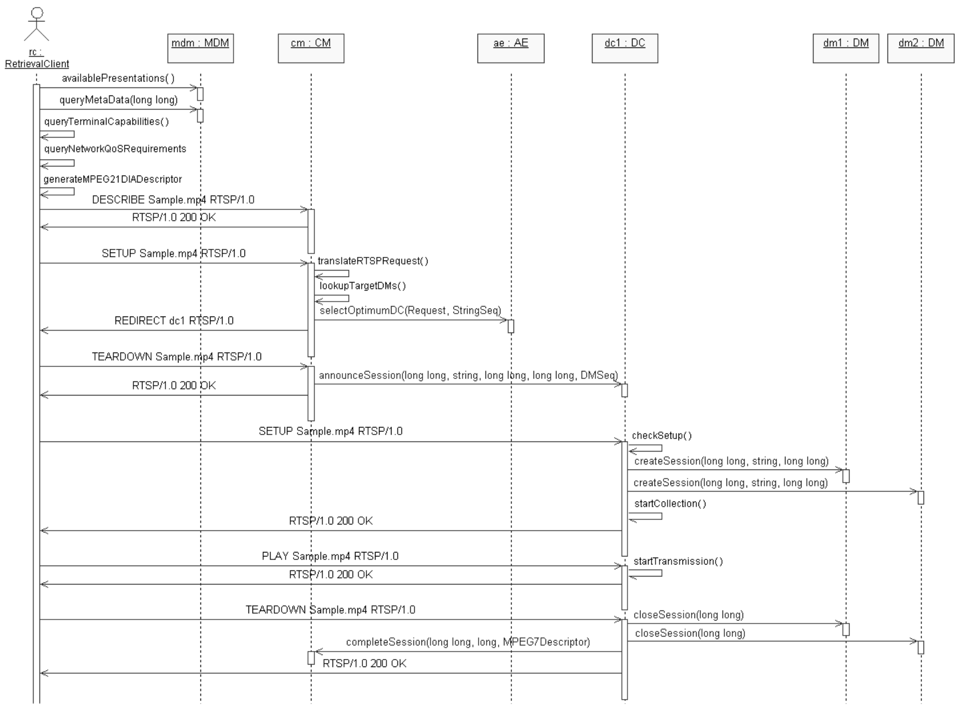

rc : RetrievalClient
mdm : MDM
cm : CM
ae : AE
dc1 : DC
dm1 : DM
dm2 : DM
availablePresentations()
queryMetaData(long long)
queryTerminalCapabilities()
queryNetworkQoSRequirements
generateMPEG21DIADescriptor
DESCRIBE Sample.mp4 RTSP/1.0
RTSP/1.0 200 OK
SETUP Sample.mp4 RTSP/1.0
translateRTSPRequest()
lookupTargetDMs()
selectOptimumDC(Request, StringSeq)
REDIRECT dc1 RTSP/1.0
TEARDOWN Sample.mp4 RTSP/1.0
announceSession(long long, string, long long, long long, DMSeq)
RTSP/1.0 200 OK
SETUP Sample.mp4 RTSP/1.0
checkSetup()
createSession(long long, string, long long)
createSession(long long, string, long long)
startCollection()
RTSP/1.0 200 OK
PLAY Sample.mp4 RTSP/1.0
startTransmission()
RTSP/1.0 200 OK
TEARDOWN Sample.mp4 RTSP/1.0
closeSession(long long)
closeSession(long long)
completeSession(long long, long, MPEG7Descriptor)
RTSP/1.0 200 OK

FIG. 2.1. *Interaction Scenario for Retrieving a Presentation from a SDMS*

A retrieval client initially queries the *mdm* for a set of available presentations on the SDMS. For a particular presentation the client asks for metadata (in MPEG-7[10] format), describing for example a temporal decomposition of the stream into segments. Afterwards, the client collects the terminal capabilities of the client device and the network QoS requirements imposed by the requested presentation. Since QoS constraints are used for data collector selection by the *ae*, they have to be communicated using a standardized descriptor. For example this can be an MPEG-21 descriptor [11, 12], as contained in the RTSP SETUP message illustrated in figure 2.2.

The MPEG-21 descriptor is rooted by the *DIA* element. The capability element of the descriptor says that

```
SETUP rtsp://sdms.itec.uni-klu.ac.at/Sample.mp4 RTSP/1.0
CSeq: 2
Range: npt=20-40;time=20030301T140000Z
Content-Type: application/mpeg21_dia
Content-Length: 557
<?xml version=”1.0” encoding=”UTF-8”?>
<DIA xmlns=”urn:mpeg:mpeg21:dia:schema:2003” xmlns:mpeg7=”urn:mpeg:mpeg7:schema:2001”
   xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
   xsi:schemaLocation=”urn:mpeg:mpeg21:dia:schema:2003 UsageEnvironment.xsd”>
   <Description xsi:type=”UsageEnvironmentType”>
      <Network>
         <Capability maxCapacity=”128000” minGuaranteed=”32000”
               inSequenceDelivery=”false” errorDelivery=”true”/>
         <Condition>
            <Utilization maximum=”1.0” average=”0.5” avgInterval=”1”/>
            <Delay packetOneWay=”500” delayVariation=”100”/>
            <Error packetLossRate=”0.02”/>
         </Condition>
      </Network>
   </Description>
</DIA>
```

FIG. 2.2. *A RTSP SETUP Message Including an MPEG-21 DIA Descriptor*

a bandwidth range between 32 and 128 kbit/sec is acceptable, packets might be delivered out of order and may be lost. The condition element specifies that the 128 kbit/sec link may be fully utilized, but on average should only get 64 kbit/sec. The packet delay from a data collector to the client should not be greater than 500 msec, and delay variation not greater than 100 msec. Finally, the packet loss rate must be below two percent.

The RTSP message further indicates that the client wants to get the stream segment from the 20th to the 40th second, relative to the media time. Additionally, the request should be scheduled for March 1st, 2003 at 14 o'clock. This time indication at session setup enables the *ae* to proactively schedule the request to an appropriate data collector, taking into account the given QoS constraints, and the locations of the required data managers. The *cm* achieves this by calling the method *selectOptimumDC(Request,StringSeq)* on the *ae*. The *ae* solves the dynamic server selection problem based on a statically configured set of data collector hosts. It achieves this by communicating with a resource broker (not illustrated in the figure), which provides load information of each server host, as well as host distance information between server-server, and server-client hosts. If there exists at least one data collector which can handle the request, the *cm* redirects the client to the recommended data collector (*dc1* in this case). The limitation of a SDMS lies exactly in the result of this method call. If the method call results in an empty set, the client request has to be rejected, due to server resource limitations. In an adaptive distributed multimedia streaming server (ADMS) environment, where a new facility (e.g. data collector) may be created on demand, it may be possible to accept the client request.

The key objective of ADMS is to maximize the number of acceptable client requests by replicating/migrating server components and media objects on demand to idle server nodes. These actions allow to minimize startup latencies and to reduce network traffic along the delivery paths to the clients. The goal can be achieved by applying a number of adaptation strategies, such as keeping server nodes load balanced, or letting clients be served by their nearest data collector. To perform these adaptation steps, a set of services has to be provided by the underlying infrastructure, which can be divided into two major classes: *application services* and *adaptation services*.

Consider figure 2.3, where an ADMS cluster C_j consists of a set of nodes M either being idle or running a certain server component. Each node is equipped with the ADMS runtime environment. Consider also 1 node being reserved for the *cluster manager*, k nodes running the *data manager* component, and l nodes running a *data collector* component ($k + l + 1 < M$ must hold). The *application service* of the underlying infrastructure has to provide means for replicating or migrating a server component from a host to one of the $M - k - l - 1$ remaining hosts, which are in an idle state. This is illustrated in figure 2.3 by replicating the *data collector* component to cluster node CN_{M-1}. Since a multimedia component may be dependent on program and data files, it also has to provide facilities for carrying those dependencies during the replication/migration processes. Migration or replication of stateful components in real-time is not required.

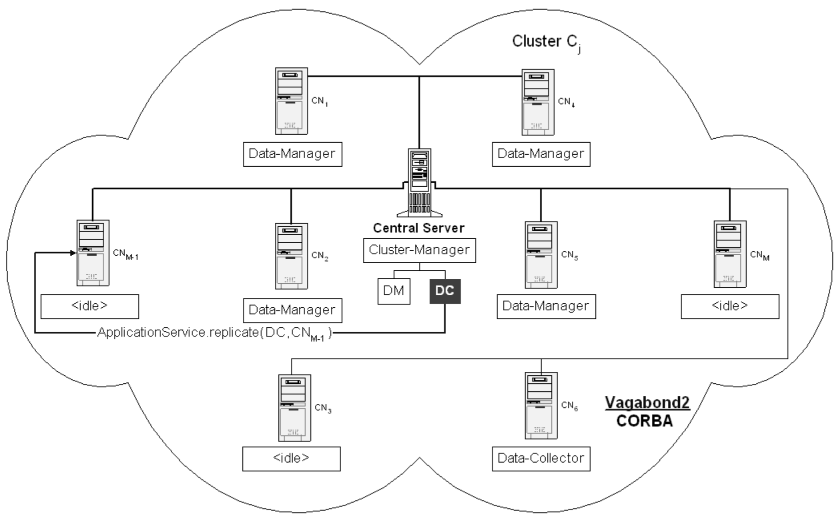


FIG. 2.3. *Replication of a Data Collector to the Idle Node CN_{M-1}*

The *adaptation services* have to provide means for automatically controlling an optimal distribution of server components, as well as host recommendations in the case of a required component adaptation. Thus, the adaptive streaming server must also be able to delegate the adaptation control over a certain component (typically the data collectors) to the underlying middleware. On the other hand, the given recommendations have to take into account server resource information (e.g. CPU, network, disk and memory usage) of each server node, global information concerning the quality of the network connections between the cluster's server nodes, as well as the topological distances and QoS parameters of the clients served by ADMS.

3. Needs by the Underlying Infrastructure. The major problem for the middleware—concerning the requirements of the adaptive server—is the one of recommending a set of server nodes for a required application-adaptation. The data collectors described in §2 are distributed in the network using the services of Vagabond2. One central component of the infrastructure, namely the so-called *host recommender*, should recommend host machines (*Harbour*s) where these applications should be loaded. In particular, the host recommender should (1) provide recommendations concerning the set of hosts that should be used by the data collectors, (2) give hints on the costs (in time) for a certain adaptation, and finally (3) dynamically adapt an already loaded application according to the previous two offers.

The problem of recommending the optimum set of server nodes can be modelled as a capacitated facility location problem (or k-median problem, where k is a given constraint). In [6] it is shown that this problem is NP-hard for general graphs. However, several approximation algorithms have been studied recently and the best published approximation factor to date is 4, with a running time of $O(N^3)$ [2].

In order to perform approximative recommendations for application locations the host recommender needs information about the server nodes, the network, as well as the current and future set of requesting clients. Concerning the server nodes the infrastructure itself can monitor the usage of the server's resources (CPU-load, disk and memory usage). Regarding the quality of the network links between the server nodes a network monitoring agent can be used to periodically perform observations on network throughput and latency times. The major requirements of Vagabond2 to the managed multimedia components are (1) the estimated resource usage of each component, (2) the topological locations of the current and future set of clients serviced by the components, and (3) the QoS-parameters of the client requests (bandwidth, delay, loss rate). Section 4 provides a more detailed specification of these requirements.

4. Interfacing ADMS and Vagabond2. Our intention when designing the interface between ADMS and Vagabond2 was to minimize the coupling between them. Some parts of the interface are implemented on the side of the media server, some on the side of Vagabond2. Either party only depends on the interface when communicating with the other one. There are three major parts of the interface definition (given in CORBA IDL syntax): *basic*, *application-specific*, and *adaptation-specific*.

The *basic part*—illustrated in figure 4.1—holds interfaces that are implemented on the media server's side. They provide information about the applications Vagabond2 should run, and the clients using these applications. Date defines a structure to hold data about a time instance. RequestInfo contains QoS requirements of a client request. TimedRequest includes a set of requests scheduled for the same date (e.g. a video and an audio elementary stream of an MPEG-4 presentation). Client defines the information to be provided about an ADMS client. Interface *ApplicationInfo* provides all required information for running and managing a server application on a remote host.

```
module vagabond2 {
  // exceptions and basic type declarations

  struct Date {
    short seconds;
    short minutes;
    short hours;
    short day;
    short month;
    long year;
  };

  struct RequestInfo {
    long long avgBitRate;
    long long maxBitRate;
    float maxLossRate;
    long maxDelay;
    long maxJitter;
    long maxLatency;
    long long duration;
  };
  typedef sequence<RequestInfo> RequestInfoSeq;

  struct TimedRequest {
    Date startDate;
    RequestInfoSeq requestInfos;
  };
  typedef sequence<TimedRequest> TimedRequestSeq;

  struct Client {
    string hostAddress;
    TimedRequestSeq timedRequests;
  };

  interface ApplicationInfo {
    string getApplicationName();
    string getMainClassName();
    OctetSeq getApplicationJAR()
      raises (ServerIOException);
    OctetSeq getDependentFilesZIP()
      raises (ServerIOException);
  };
  typedef sequence<Client> ClientSeq;

  interface AdaptiveApplication {
    void start(in StringSeq params)
      raises (ServiceAlreadyStartedException);
    void stop()
      raises (ServiceNotStartedException);
    boolean isIdle();
    ApplicationInfo getApplicationInfo();
    ClientSeq getClients();
    void setLocked(in boolean locked);
    boolean isLocked();
  };
}; // vagabond2
```

FIG. 4.1. *The Basic Part of Vagabond2's CORBA-based Interface*

The interfaces in the *application specific part* are implemented in Vagabond2. They provide services for loading, evacuating and locating adaptive server applications. Interface *ApplicationLocator* provides means for locating all hosts on which a certain application currently runs on. Each application that has been loaded on at least one host has an own application locator. Interface *ApplicationService* provides means for loading a given application on a given host, evacuating an application from a given host, locating server application instances, and retrieving all applications on a given host. It is the centre of the application part, whose IDL specification is shown in figure 4.2.

The interfaces of the *adaptation specific part*—presented in figure 4.3—are implemented in Vagabond2. They provide services for adapting applications loaded by Vagabond2 with respect to an optimal distribution. Interface *HostRecommender* is responsible for recommending a set of hosts for a certain application by using topological and resource usage information. It also checks whether an application has optimal distribution. AdaptationPolicy represents an adaptation policy to be used by the adaptation service. Interface *AdaptationService* provides access to objects being responsible for and dealing with application-adaptational issues. It can return a host recommender, may suggest a policy by taking into account the data of clients, and can adapt an application by reorganizing its distribution.

We are currently working on adaptation strategies based on statistical information about the nodes and network. This information is collected by a separate part of the system, called ResourceBroker. We have recently tested a greedy algorithm that proved that the TCP-based connections between data collectors and data managers should be regarded differently from the RTP/UDP-based connections between the data collectors and the clients. We are now testing how different weights on different link-properties can improve the adaptation process of the media server architecture.

```
module vagabond2 {
  module services {
    module management {
      module application {

        interface ApplicationLocator {
          string getApplicationName();
          StringSeq getLocations();
          AdaptiveApplication getApplicationObject(in string hostAddreess)
            raises (NoSuchHostException);
        };

        interface ApplicationService {
          boolean load(in vagabond2::ApplicationInfo ai, in string hostAddress)
            raises (ApplicationAlreadyLoadedException, NoSuchHostException);
          boolean evacuate(in string appName, in string hostAddress,
            in boolean force)
            raises (NoSuchApplicationException, ApplicationNotLoadedException,
              NoSuchHostException, ApplicationNotIdleException);
          StringSeq getApplications(in string hostAddress)
            raises (NoSuchHostException);
          ApplicationLocator getApplicationLocator(in string appName)
            raises (NoSuchApplicationException);
          boolean runsOn(in string appName, in string hostAddress)
            raises (NoSuchApplicationException, NoSuchHostException);
        };
      }; // application
    }; // management
  }; // services
}; // vagabond2
```

FIG. 4.2. *The Application Module of Vagabond2*

5. The Architecture of Vagabond2. Vagabond2 is based on Vagabond, a mobile agent system developed at the Budapest University of Technology and Economics [4]. It is a CORBA-based[13] mobile agent system written in Java and was originally developed to be used in multimedia database environments [5]. Vagabond2 is also CORBA-based and written in Java. It is able to migrate CORBA objects, and pass their references to their clients. It keeps track of the available server hosts, the applications they are running and the CORBA objects that incarnate the applications. These functionalities are hidden behind the interfaces described in §4.

Figure 5.1 illustrates the implementation class hierarchy of Vagabond2. For application/component management two public interfaces are provided: AppService and AppLoc ("Application" is abbreviated to "App", "Locator" to "Loc" in this section). The system consists of server hosts that are able to load and run Java-CORBA objects, and applications that run on some of the servers. An application therefore consists of several CORBA objects on different hosts, where the objects can be thought of as 'incarnations' of the application.

The two major responsibilities are (1) providing the server hosts that can load Java objects and publish them as CORBA objects, and (2) maintaining the information of the relations among applications, their incarnations, and server hosts. The former required a simple modification of the Vagabond system. We defined a new interface *Harbour*, which will take care of this responsibility. Concerning the second responsibility it is clear that there are one-to-many relations between an application and its incarnations, and between a host and the incarnations running on it. For the sake of efficiency we decided to introduce a set of redundant relations: {application, hosts}, {application, incarnations}, {host, applications}.

The *Harbour* CORBA interface defines the methods provided by objects that are able to receive, run, and evacuate Java objects, and connect them to the ORB. It represents the runtime environment for adaptive server applications. Every server host must run a Harbour. Recently new extensions were added to this interface. We have defined several time points when migrating an application: download, local storing, class loading and object incarnation times. A Vagabond2 client can access these times for every server application loaded on a Harbour. Furthermore, host profile information can be obtained about the host on which a Harbour is running.

The classes starting with "Host" are responsible for storing the information about available server hosts. A HostData object stores information about a single host: a reference to an object implementing the *Har-*

```
module vagabond2 {
  module services {
    module management {
      module adaptation {
        interface HostRecommender {
          boolean hasOptimalDistribution(in string appName,
            in ClientSeq clients);
          StringSeq recommendHosts(in ApplicationInfo ai,
            in ClientSeq clients);
        };

        enum AdaptationPolicy {
          MINIMUM_STARTUP_DELAY,
          LOAD_BALANCED,
          MINIMUM_NETWORK_CONSUMPTION
        };

        interface AdaptationService {
          HostRecommender getHostRecommender();
          AdaptationPolicy getAdaptationPolicy();
          AdaptationPolicy suggestPolicy();
          StringSeq getAvailableHosts(in ApplicationInfo ai);
          void adaptApplication(in string appName)
            raises (NoSuchApplicationException);
          void releaseApplication(in string appName)
            raises (NoSuchApplicationException);
          StringSeq getAdaptedApplications();
        };
      }; // adaptation
    }; // management
  }; // services
}; // vagabond2
```

FIG. 4.3. *The Adaptation Module of Vagabond2*

bour CORBA interface, and the names of the applications currently running on that host. HostStore manages {hostname, HostData} couples. *HostService* provides a CORBA interface for updating the HostStore remotely.

The AppData objects store the {hostname, *Object*} couples for a given application, where the Object is a CORBA object that was loaded on the given host. It also stores a reference to the *AppInfo* object that describes the given application. AppStore stores {appname, AppData} couples.

The AppService_impl implements the *AppService* CORBA interface, has access to the HostStore and AppStore objects, and has a list of objects implementing the *AppLoc* CORBA interface for every loaded application. It will load and evacuate the objects incarnating an application on Harbours that are registered by *HostService*. The load() and evacuate() methods (1) manage the information about newly loaded or removed objects in HostStore and AppStore; (2) call the load() and evacuate() methods of the Harbour object concerned.

6. Proof of Concept: Measurements. We evaluated the system in a testbed shown in figure 6.1. The testbed consists of eight servers in two LANs (B-LAN, I-LAN), interconnected via the Internet. The B-LAN is located in Budapest (Hungary), and the I-LAN in Klagenfurt (Austria). The data collector runs on a client in the I-LAN. Each performance test consists of five test runs and is repeated 50 times. In each run, a sample media stream of 12,5 MB size is retrieved from the data manager instances. In test run 0 all data manager instances are running in the remote B-LAN. In test run 1 the data manager from host 8 is replicated to host 4, meaning that one data manager has been moved closer to the data collector. In particular both, the component code and the requested media stream are replicated, since hosts 1—4 are in an idle state initially. In each further test run one additional data manager is replicated from a host in the B-LAN to a host in the I-LAN.

Figure 6.2 illustrates error plots of the mean retrieval times and throughputs for the test runs in the test scenario. It clearly shows that the more data managers are replicated to the I-LAN, the shorter the retrieval times, and the higher the throughput become. Whereas the variations of retrieval times in the B-LAN are quite high, they get much less in the I-LAN.

An interesting property of data managers can be derived from figure 6.3. It tells the relative gain on

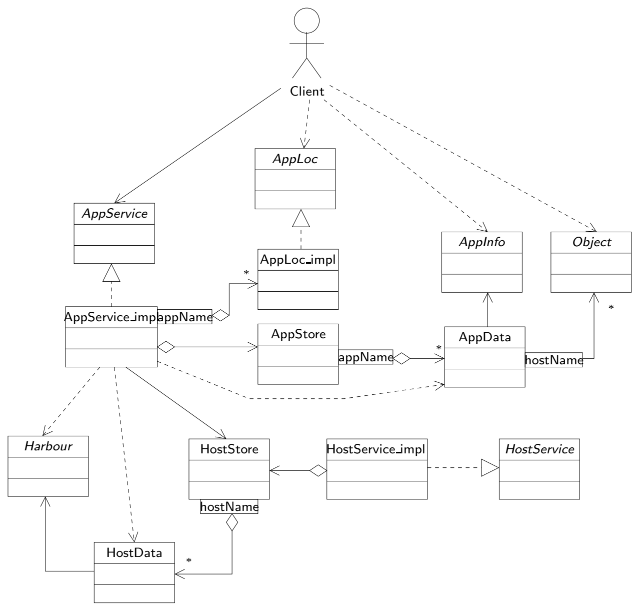
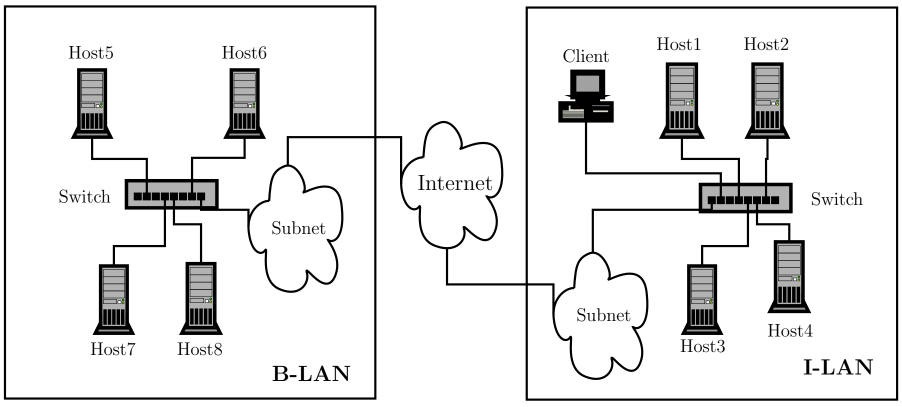


FIG. 5.1. *Implementation Class Hierarchy*

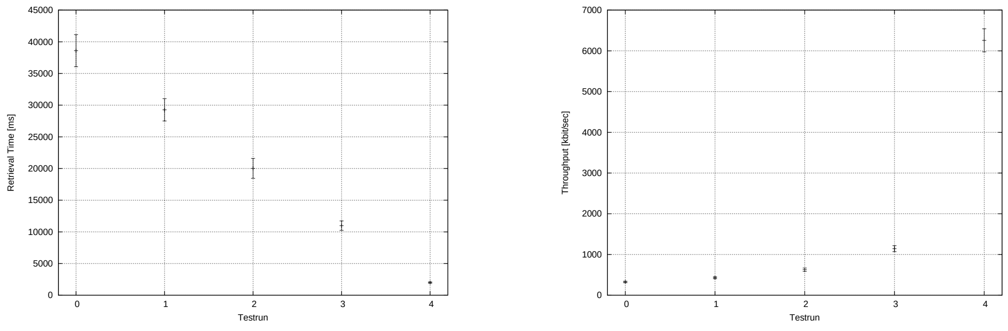


Testrun	DM Hosts
0	Host5, Host6, Host7, Host8
1	Host5, Host6, Host7, Host4
2	Host5, Host6, Host3, Host4
3	Host5, Host2, Host3, Host4
4	Host1, Host2, Host3, Host4

FIG. 6.1. *The Testbed Setup*

throughput, if a certain amount of stripe units is replicated from one LAN to another (including the time for component code replication), based on a simple model[1]. We can see that as the maximum throughput gain increases, the elbow of the curves gets closer to the bottom right corner. This means, that if we want to have a significant increase in relative gain, we have to replicate a high percentage of data managers—usually almost all of them. Thus, the best choice is to keep them together.

[1]Let t_f and t_c be throughputs when all the managers are on the farther LAN, or the closer LAN, respectively. The maximum gain is the ratio of them, $g_m = t_c/t_f$. Let t_x be the throughput, if x part of the managers are on the closer LAN. The relative gain is $g_r = t_x/t_c$. The model states that the relative gain is a function of x and g_m: $g_r(x, g_m) = \frac{1}{x+(1-x)g_m}$. In the case of the measurement, g_m was about 20.



(a) Mean stream retrieval time (b) Mean throughput

FIG. 6.2. *Performance of test runs in the test scenario*

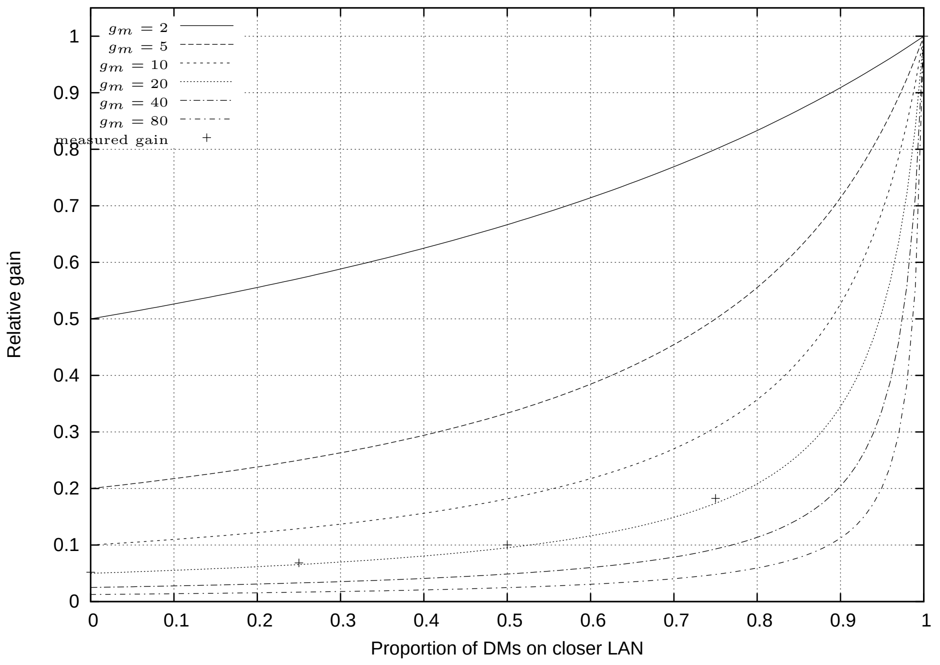


FIG. 6.3. *The results of the model and the measurements*

7. Conclusion and Future Work. We addressed the requirements of the adaptive distributed multimedia streaming server architecture *ADMS* to its underlying infrastructure Vagabond2 and vice versa, regarding the management and adaptation of multimedia components. From the point of the server the middleware has to provide two major classes of services, namely *application* and *adaptation* services. On the other hand, for Vagabond2 performing its tasks of managing applications and recommending hosts, the server has to provide detailed information on the applications to manage, as well as on the clients requesting services from those applications (i.e. information on location and request parameters). This set of bilateral requirements resulted in the specification of a set of CORBA-based interfaces, which all together define the interface between ADMS and Vagabond2. Using this interfaces Vagabond2 provides means for (1) loading and starting CORBA components written in Java on any host that runs Vagabond2, (2) evacuating a component, and (3) querying the location and distribution of components. All of these services can be accessed by CORBA method invocations.

We proved our concept by evaluating the replication of data managers between two LANs interconnected by the Internet. The results have shown that it is recommended to keep the data managers as close to each other, as possible. We are currently working on a support for adaptation using the knowledge gained from the network topology, the load of links and hosts, and the needs of the clients. This knowledge is used to implement different adaptation strategies in the host recommender. Recently a greedy algorithm was tested with promising results. In this test we realized that a distinction has to be made when considering the TCP-based connections between data collectors and data managers, and the RTP/UDP-based connections between data collectors and clients.

A resource broker subsystem is already working and gives the adaptation service the information about the statistical properties of server nodes and networks. Furthermore, a complete implementation and detailed evaluation of a distributed adaptive multimedia streaming application based on MPEG4-encoded media streams is planned, using Vagabond2 as the infrastructure for performing adaptational issues. In particular, it should be figured out which adaptation strategy performs best under a given set of constraints (e.g. the application distribution and the request parameters). Moreover, it should be investigated whether Vagabond2 would be able to perform *on the fly* migration or replication of server components, and under which restrictions this would be the case.

REFERENCES

[1] A. Bieszczad, B. Pagurek, and T. White, *Mobile Agents for Network Management*, IEEE Communication Surveys, 1 (1998), pp. 2–9.

[2] M. Charikar and S. Guha, *Improved Combinatorial Algorithms for the Facility Location and k-median Problems*, in IEEE Symposium on Foundations of Computer Science, 1999, pp. 378–388.

[3] R. Friedman, E. Biham, A. Itzkovitz, and A. Schuster, *Symphony: An Infrastructure for Managing Virtual Servers*, Cluster Computing, 4 (2001), pp. 221–233.

[4] B. Goldschmidt and Z. László, *Vagabond: A CORBA-based Mobile Agent System*, in Object-Oriented Technology ECOOP Workshop Reader, A. Frohner, ed., Springer Verlag, 2001.

[5] B. Goldschmidt, Z. László, M. Döller, and H. Kosch, *Mobile Agents in a Distributed Heterogeneous Database System*, in Euromicro Workshop on Parallel, Distributed and Network-based Processing, 2002.

[6] O. Kariv and S. L. Hakimi, *An Algorithmic Approach to Network Location Problems. II: The p-medians*, SIAM Journal of Applied Mathematics, 37 (1979), pp. 539–560.

[7] J. Y. Lee, *Parallel Video Servers: A Tutorial*, IEEE Multimedia, 5 (1998), pp. 20–28.

[8] B. Li and K. Nahrstedt, *A Control-based Middleware Framework for Quality of Service Adaptations*, IEEE Journal of Selected Areas in Communications, (1999), pp. 17(9):1632–1650.

[9] Moving Pictures Experts Group, *ISO/IEC JTC1/SC29/WG11 N4668: Overview of the MPEG-4 Standard*, 2002. `http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm`

[10] ———, *ISO/IEC JTC1/SC29/WG11 N4980: MPEG-7 Overview*, 2002. `http://mpeg.telecomitalialab.com/standards/mpeg-7/mpeg-7.htm`

[11] ———, *ISO/IEC JTC1/SC29/WG11 N5231: MPEG-21 Overview*, 2002. `http://mpeg.telecomitalialab.com/standards/mpeg-21/mpeg-21.htm`

[12] ———, *ISO/IEC JTC 1/SC 29 N 5204: Text of ISO/IEC 21000-7 CD - Part 7: Digital Item Adaptation*, 2003. `http://www.itscj.ipsj.or.jp/sc29/open/29view/29n5204c.htm`

[13] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.6 ed., December 2001.

[14] J. Rolia, S. Singhal, and R. Friedrich, *Adaptive Internet Data Centers*, SSGRR'00, (2000).

[15] R. Tusch, *Towards an Adaptive Distributed Multimedia Streaming Server Architecture Based on Service-oriented Components*, Tech. Report TR/ITEC/03/2.02, Institute of Information Technology, Klagenfurt University, Klagenfurt, Austria, February 2003. Paper submitted to the Joint Modular Languages Conference (JMLC) 2003.

[16] R. Tusch, L. Böszörményi, B. Goldschmidt, H. Hellwagner, and P. Schojer, *Offensive and Defensive Adaptation in Distributed Multimedia Systems*, Tech. Report TR/ITEC/03/2.03, Institute of Information Technology, Klagenfurt University, Klagenfurt, Austria, February 2003. Paper submitted to the Journal of Systems and Software, Special Issue on Multimedia Adaptation.

[17] J. Waldo, *The Jini Architecture for Network-centric Computing*, Communications of the ACM, 42 (1999), pp. 76–82.

Edited by: Zsolt Nemeth, Dieter Kranzlmuller, Peter Kacsuk, Jens Volkert
Received: March 24, 2003
Accepted: June 5, 2003

Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 93–104. http://www.scpe.org
ISSN 1895-1767
© 2005 SWPS

AN ADAPTIVE STANDARD META-DATA AWARE PROXY CACHE

PETER SCHOJER, LASZLO BÖSZÖRMÉNYI AND HERMANN HELLWAGNER*

Abstract. Multimedia is gaining ever more importance on the Internet. This increases the need for intelligent and efficient video caches. A promising approach to improve caching efficiency is to adapt videos. With the availability of MPEG-4 it is possible to develop a standard compliant proxy cache that allows fast and efficient adaptation.

We propose a modular design for an adaptive MPEG-4 video proxy that supports efficient full and partial video caching in combination with filtering options that are driven by the terminal capabilities of the client. We use the native scalability operations provided by MPEG-4, the MPEG-7 standard to describe the scalability options for a video and the emerging MPEG-21 standard to describe the terminal capabilities. In this paper, we will restrict ourselves to full video caching.

The combination of adaptation with MPEG-4, MPEG-7 and client terminal capabilities is to the best of our knowledge unique and will increase the quality of service for end users.

Key words. Adaptation, MPEG-4, MPEG-7, MPEG-21, adaptive proxy, caching

1. Motivation. The importance of multimedia on the Internet is steadily increasing. A new dimension of complexity is emerging by the heterogeneity found in end terminals. Adaptation can be used in Web proxies to improve caching efficiency and to support new end terminals. On the one hand, similarly to usual Web proxy caching, storing popular videos close to the clients will reduce network/server load and start-up latency. On the other hand, the proxy can be used as a media gateway to *adapt* videos to fit the needs of a client.

In this paper we show the design of an adaptive Web proxy using RTSP and the multimedia standards MPEG-4, MPEG-7 and MPEG-21.

2. Related Work. There has been few practical work carried out in the area of adaptation of cached, layered encoded videos.

In [6] an adaptive proxy is introduced. The corresponding implementation is based on a proprietary AT&T client-server architecture, where little public information is available. [8] proposes a proxy that filters GOPs (Group of Pictures) based on the available bandwidth; only simulation results are presented. [9] proposes a way to deal with retransmission of missing segments of layered encoded videos; again simulations are used to derive results. [5] performs an exhaustive comparison of replacement strategies suitable for layered encoded videos, again based on simulation results. [7] proposes a pre-fetching mechanism for optimizing the quality hit rate in a proxy (a metric considering the quality of an adapted video compared to the quality of the original video).

While there is nothing wrong with simulation when evaluating new video cache replacement strategies, simulation reaches its limits if one wants to analyze how a CPU intensive task like an adaptation step effects the overall performance/scalability of the proxy.

To the best of our knowledge, we are actually the first who present a design and an implementation (still in progress) for an adaptive MPEG-4 proxy that considers terminal capabilities and MPEG-7 meta-information.

3. Background.

3.1. Adaptation.

Media adaptation is the ability of a system to change the quality of a multimedia stream according to available resources [10]. In this work, we restrict adaptation to non-linear quality reduction. Non-linearity in this context means that we gain a large size reduction for a relatively moderate quality loss.

In [5] several cache replacement strategies for layered videos have been evaluated, clearly showing that using adaptation in proxies not only results in a higher object hit rate but also in a higher quality-weighted hit rate (a metric combining object hit rate and quality hit rate).

A special variant of adaptive proxies are *media gateways* that allow low-end devices to view a down-scaled version of the video. Media gateways additionally support transcoding of videos, a CPU intensive task.

The integration of terminal capabilities and meta-information enables our proxy to do both adaptation for the sake of a higher hit rate and to act as a simple media gateway, doing efficient adaptation in the compressed domain.

As already indicated previously, adaptation steps depend on the type of videos used, e.g., resizing MPEG-1 videos requires CPU intensive de- and recompression of the video. Fortunately, with MPEG-4 a standard is available that provides support for adaptation.

*University of Klagenfurt, Institute of Information Technology ({pschojer, laszlo, hellwagn}@itec.uni-klu.ac.at)

3.2. Object-based Coding in MPEG-4. MPEG-4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group). It became an International Standard in the first months of 1999. Compared to older MPEG standards, it is the first one that offers the possibility to structure audio-visual streams. MPEG-4 defines the notion of a scene (see Figure 3.1) relying on similar notions of virtual reality standards. A scene consists of a set of media objects, each having a position, a size and a shape [1].

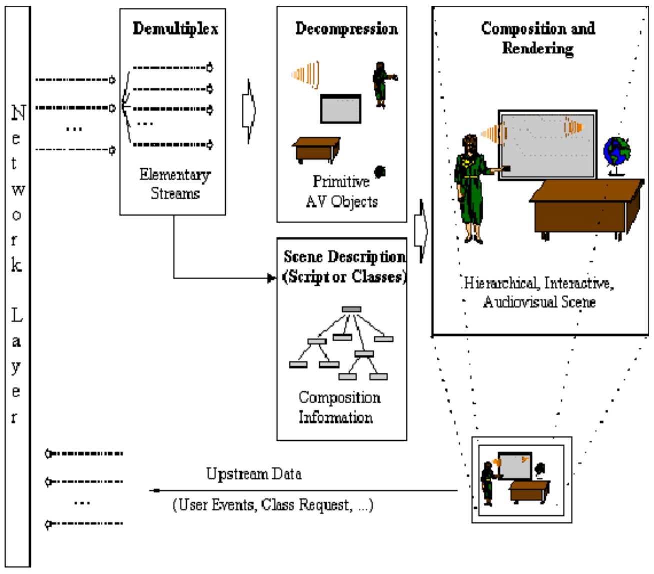

Network Layer
Demultiplex
Elementary Streams
Decompression
Primitive AV Objects
Scene Description (Script or Classes)
Composition Information
Composition and Rendering
Hierarchical, Interactive, Audiovisual Scene
Upstream Data
(User Events, Class Request, ...)

Fig. 3.1. *Example of an MPEG-4 scene [1]*

Each media object consists of at least one *Elementary Stream* (ES). If more than one ES is present for a visual object, the object offers native adaptation support. The additional ESs are used to add spatial, temporal, or SNR scalability to an object. Fast adaptation can be done by simply dropping an ES, which is equivalent to deleting a file in the proxy.

An ES itself consists of a sequence of *Access Units* (AUs), where usually one AU encapsulates one frame. To transfer the AU over the network, the AU is packetized into *SyncLayer packets* and then passed to the delivery layer, where it is packetized into an RTP packet. Depending on the size of the AU, one or more SL packets are needed [1].

An MPEG-4 movie (with video and audio) consists of at least five ESs. One is the *Initial Object Descriptor* (IOD) stream that keeps references to the objects used in the video. The objects are described in the *Object Descriptor* (OD) stream. The third mandatory stream is the BIFS stream (Binary Information For Scenes), that stores the scene description, actually telling the decoder which object is visible at which point in time and where exactly it should be rendered. Optionally, one can have several ESs for each object [1].

While MPEG-4 offers support for adaptation, the question remains how the proxy determines which adaptation step will give the most benefit in a certain situation. This can only be solved by adding meta-information to a video, as defined by MPEG-7.

3.3. Multimedia Meta-data. MPEG-7 is another ISO/IEC standard and aims at describing multimedia data. With MPEG-7 one can add semantic information to a video. For the proxy, meta-information regarding adaptation is especially important. We restrict ourselves to the content adaptation part of MPEG-7; for further details see [2].

MPEG-7 features a *Description Definition Language* (DDL), that allows one to generate a *Description Schema* (DS), which in turn is used to code *Descriptors*.

The VariationRelationship descriptor of the Variation DS is used to describe a possible adaptation step. There are several descriptors defined, like *Summarization, Extraction, ColorReduction, SpatialReduction*, or *QualityReduction*, to name just a few.

A simplified MPEG-7 description containing an adaptation sequence (=*VariationSet*) consisting of two variations might look like this:

```
<Description xsi:type="VariationDescriptionType">
  <VariationSet>
    <Source xsi:type="VideoType"> [..]
      <ComponentMediaProfile id="ES1">  [..]
      <ComponentMediaProfile id="ES2">  [..]
      <MediaFormat>
        <Content href="MPEG7ContentCS">
          <Name>audiovisual</Name>
        </Content>
        <FileFormat
          href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5">
          <Name xml:lang="en">mp4</Name>
        </FileFormat>
        <FileSize>13107200</FileSize>
        <BitRate>131072</BitRate>
      </MediaFormat>
    [..]
    </Source>
    <Variation id="VARIATION1"
               fidelity="0.75"
               priority="1"> [..]
    <Variation id="VARIATION2"
               fidelity="0.45"
               priority="2"> [..]
  </VariationSet>
</Description>
```

The “source part” describes the internal structure of the MPEG-4 video. For each ES it contains one *ComponentMediaProfile* description, for the complete video it contains one *MediaFormat* block that describes the total size of the video in bytes and the average bit rate of the video, i.e., the size is 12.5 MByte and the bit rate is 128 kBit/sec.

The following example describes in detail one ES of type “visual” with a dimension of 352x288 and a frame rate of 30 frames per second:

```
<ComponentMediaProfile id="ES1">
  <MediaFormat>
    <Content href="MPEG7ContentCS">
      <Name>visual</Name>
    </Content>
    <VisualCoding>
      <Format href="urn:mpeg:mpeg7:cs:\dots  [..]"/>
      <Name xml:lang="en">MPEG-4 Visual</Name>
      <Frame height="288" width="352" rate="30.00"/>
    </VisualCoding>
  </MediaFormat>
</ComponentMediaProfile>
```

The source part is followed by the description of the available variations. A variation description is shown in the next MPEG-7 fragment. The effects of a TemporalReductionAdaptor on the video are described.

```
<Variation id="VARIATION1"
  fidelity="0.75"
  priority="1">
[..]
  <ComponentMediaProfile id="ES1">
   <MediaFormat>
    <Content href="MPEG7ContentCS">
```

```
      <Name>visual</Name>
    </Content>
    <VisualCoding>
      <Format href="urn:mpeg:mpeg7:cs:\dots  [..]"/>
      <Name xml:lang="en">MPEG-4 Visual</Name>
      <Frame height="288" width="352" rate="7.50"/>
    </VisualCoding>
   </MediaFormat>
  </ComponentMediaProfile>
  <ComponentMediaProfile id="ES2"> [..] </ComponentMediaProfile>
  <MediaFormat> [..]
    <FileFormat [..]
    <FileSize>6553600</FileSize>
    <BitRate>65536</BitRate>
  </MediaFormat>
[..]
  <VariationRelationship> TemporalReduction
  </VariationRelationship>
</Variation>
```

The proxy knows from this description that applying this adaptor will decrease the frame rate down to 7.5 frames, which for this specific video results in a quality loss of only 25% (*fidelity="0.75"*). The proxy also knows that 6.25 MBytes are gained and the bit rate is halved for the video. The priority field is used by the proxy to sort variations within a VariationSet. Unfortunately, the current version of MPEG-7 doesn't support the specification of resource consumptions [4]. But typically, the proxy can estimate how much CPU cycles a variation will use, so this missing feature is not fatal.

3.4. MPEG-21—A Multimedia Framework. MPEG-21 [3] is the latest ISO/IEC development, parts of which will become International Standard in 2004. The goal of MPEG-21 is to define a normative open framework for multimedia delivery and consumption, spanning the whole delivery/consumption chain. We are especially interested in MPEG-21 because it allows clients to declare descriptors for expressing their usage environment. In particular, Part 7—*Digital Item Adaptation* (DIA) [14]—allows specifying terminal capabilities as well as network capability and condition descriptors.

An example of using the terminal capabilities provided by MPEG-21 will be given in Section 6.

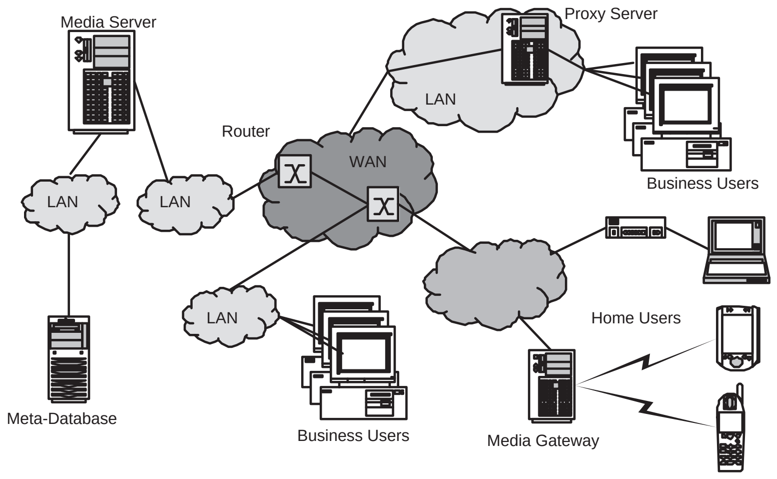


Fig. 3.2. *Components of a distributed multimedia system [10]*

4. The ADMITS Project. The adaptive proxy itself is only one part of an end-to-end adaptation chain. Figure 3.2 shows the layout of a distributed multimedia system. Adaptation options in such a system are investigated in the ADMITS project (*Adaptation in Distributed Multimedia IT Systems*) at our institution [10].

A distributed multimedia system comprises several types of components, such as media servers, meta-databases, proxies, media gateways, routers and clients. Servers, proxies and gateways—being the most powerful components in this scenario—can apply complex adaptation steps, i.e., transcoding. A router can be extended to employ, in case of congestion, a simple packet dropping scheme like dropping B-frames before P- and I-frames. Clients can also employ adaptations like dropping B-frames in case they are unable to decode the full video in real-time, though it is preferable to detect this performance gap at the sender side and not to send the data. In addition to media adaptation, the components of the multimedia system itself can be adapted; one example is server/proxy replication. The main question is, which kind of component can be best used for which type of adaptation.

4.1. Meta-Database. The meta-database supports all kinds of adaptations by supplying meta-data for the adaptation engines. In addition, the meta-database acts for the user as an entry point to the distributed multimedia architecture. It is implemented as an MPEG-7 Multimedia Data Cartridge relying on cartridge technology of the Oracle database management system (DBMS) [15]. The meta-data enables multiple functionalities:

- Support of content-based queries, based on low- and/or high-level indexing information. Low-level indexing favors similarity search, such as, "give me all clips with key frames of similar color distribution as the query image". A high-level query is, e.g., "give me all soccer clips where the Austrian player Vastic scores a goal from a free kick". The meta-database transforms the clients' queries into a sequence of video identifiers and transmits those to the video server, which in turn delivers the videos to the client.
- Support of adaptation based on meta-information, such as the definition of transcoding procedures with the help of MPEG-7 variation descriptors [10].

4.2. Adaptive Router. The operations that a router can perform are fairly limited, since forwarding speed must not be compromised significantly. The advantage is that a router is the best place to cope with varying network load (congestion). The simplest and thus fastest solution is to prioritize frames of a video. In other words, in case of congestion all packets that belong to B-frames are dropped before those belonging to P-frames; I-frames are never discarded, if possible. A new network protocol was developed that provides this prioritization in a way that allows efficient adaptation [16].

4.3. Adaptive Virtual Video Server. The Virtual Video Server (or Adaptive Multimedia Server) is a distributed video server which can change its number of nodes on-the-fly. Videos are stored as a sequence of shots striped over the nodes of the server. For example, if the server detects that a client is far away from the source, it can push the proxy code to a node which resides closer to the client and use this node as a data collector for the requested video [17].

5. Adaptive Proxy.

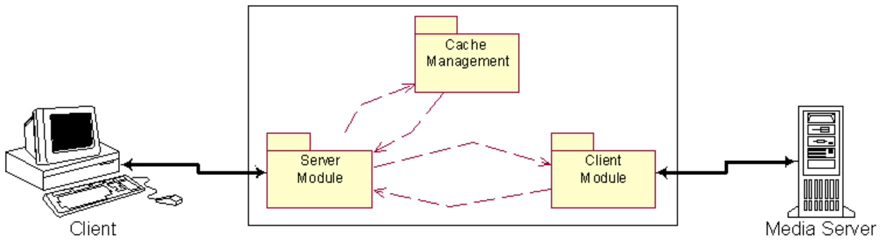


FIG. 5.1. *Proxy architecture*

5.1. Design. A proxy is inserted into the path between client and server. It has to "imitate" a server for the client and a client for the server. Thus, the design of the proxy consists of three main parts: a server module, that implements server functionality dealing with client requests; a client module that is responsible for establishing a connection with the media server; and a cache manager that manages cached objects and realizes cache replacement strategies. Figure 5.1 shows how these parts communicate with each other. The server part uses the cache manager to look up a video and to insert videos into the cache. The client part is used to retrieve missing videos/ESs from the server.

5.2. Server Module. The server module listens for RTSP requests and manages existing sessions. For a new request a *ServerSession* object is created. If the proxy encounters a partial hit, i.e., it has to fetch missing ESs from the media server, a *ClientSession* object is needed too. Each session object listens at a TCP socket for RTSP commands and forwards them to their corresponding *DataChannels*. A DataChannel is the link between the media data and the client and encapsulates an *Adaptor*. An Adaptor reads data from a *Reader*, which could be a network or a file reader, adapts internally the data—for example a TemporalAdaptor could drop all packets transporting B-frames—and then outputs the result to one or more *Writer* objects. In the case of a cache miss for an ES, the ClientSession object will create an Adaptor with a network Reader and two Writer objects: a file Writer for caching and a network Writer for forwarding. A simplified data flow is shown in Figure 5.2. To keep the example concise, the data flows writing the data to the disk are omitted.

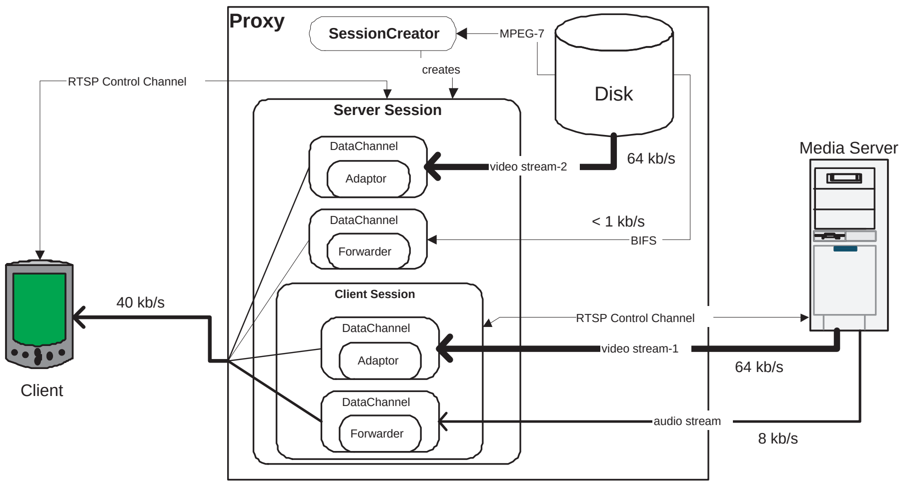


Fig. 5.2. *Data flow with adaptation*

5.3. Adaptation in the Proxy. Adaptation is part of the Server module. It supports the following:

- System level adaptation
  An MPEG-4 video can consist of several video objects. This allows for object based scalability (dropping video objects) as shown in Figure 5.3. Furthermore, each object can consist of several ESs. Having more than one ES per video object adds native support for *temporal*, *spatial* or *SNR* scalability—if the media supports it and has been encoded by the content provider to offer enhancement layers or objects. The decision to apply system level adaptation happens during the RTSP DESCRIBE phase, where an SDP (Session Description Protocol) description is generated, which contains the IOD stream and a simple description of all ESs of a video (like average bit rate of a stream, its stream-id, and whether a stream is audio or video or a generic stream like BIFS).
- ES adaptation
  Whether to adapt an ES, is decided during session creation where the Adaptors are set at the Data-Channels. If no adaptation is required, a *Forwarder* object is created that simply forwards data from the Reader to the Writers. This decision depends on the terminal capabilities and the MPEG-7 meta-information associated with the MPEG-4 stream. Adaptors work on a single ES. Currently, the following adaptation classes are supported:
  - Temporal scaling (B-frame dropping, GOP dropping, extraction of key-frames)
  - Color reduction (grey-scaling)
  - Spatial reduction
  - Bit rate reduction.

  The temporal scaling adaptors work in the compressed domain. All other operations require CPU intensive de- and recompression. More information on the adaptation process and benchmarks on the CPU load generated by one adaptation step can be found in [11].

5.3.1. Meta-Information. To adapt a video efficiently, in-depth information about the video is needed, such as which ES is enhancement layer (and thus can be removed) and which is base layer.

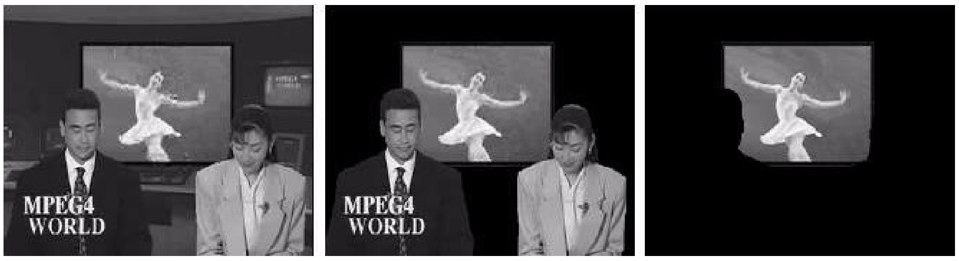


FIG. 5.3. *Adaptation on system level: object based adaptation (from [12])*

This information can be extracted from the OD stream and is needed rather early during the RTSP DESCRIBE phase. In case of a complete cache miss, one has to download the (very small) OD stream completely and parse this stream to detect these dependencies. The SDP description received from the server can be used to determine the bit rate of an individual stream and hide ESs from the client accordingly, simply by removing them from the SDP description. This is only a rough and defensive estimation based on average bit rate values. To determine the total bandwidth requirement, the BIFS stream must be parsed—but this intelligence is better integrated at the client.

To determine if an adaptation step is allowed, one has to parse the MPEG-7 stream for VariationRelationship descriptors and pick the one that best meets the requirements of the client. This stream is also needed in advance.

Additionally, the meta-information has to be updated to reflect the changes (lower bit rate, lower frame rate, update/delete variation descriptors), to avoid that another proxy applies the same adaptation. More information on the use of meta-data can be found in [10].

5.4. Cache Management. The main functionality of the cache management module is to store the received data and to realize cache replacement algorithms. Within the proxy, an ES is modelled as a sequence

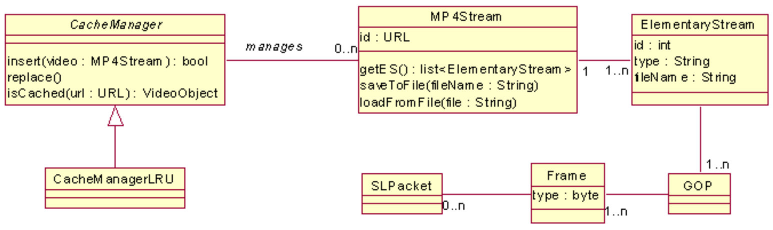


FIG. 5.4. *Cache management*

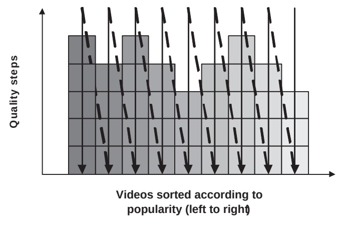
of GOPs, which consist of frames, which are stored as a sequence of SyncLayer [1] packets (Figure 5.4). The *CacheManager* class itself is defined as an abstract class, allowing several cache replacement strategies to be realized.

5.4.1. Cache Replacement. Several cache replacement strategies (CRSs) are offered by the proxy. A classical LRU was implemented for comparison reasons and more advanced variants of LRU that allow lowering the quality of a cached video in discrete steps.

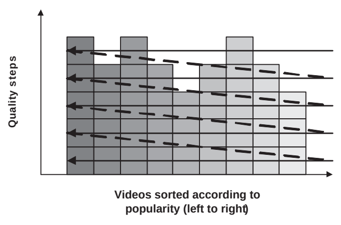
Horizontal and vertical cache replacement is supported [13]. The vertical CRS (v-CRS) successively chooses quality variations of the least popular video in the list for replacement. In the worst case, v-CRS degenerates to a classic LRU, especially when many adaptation steps have to be performed to free up enough space disk for one video. As shown in [13], the object hit-rate (hits/requests) is nearly the same when compared to LRU.

The horizontal CRS (h-CRS) chooses the adaptation candidates according to their quality. The video with the highest available quality layer is searched and adapted. This strategy suffers from a different problem. While h-CRS has a high object hit-rate, its quality hit-rate (average quality of all hits) is low. In the long run, it tends to adapt every object down to its lowest quality layer, even newly inserted videos [13].

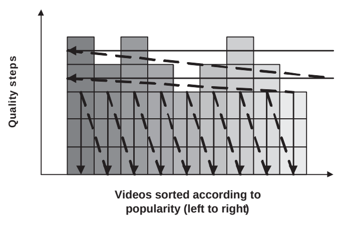
To overcome the disadvantages of these two extreme adaptation strategies, a third approach was integrated that combines vertical and horizontal CRSs as illustrated in Figure 5.5.



(a) Vertical CRS



(b) Horizontal CRS



(c) Combined CRS

FIG. 5.5. *Cache replacement strategies [13]*

6. Example. Figure 6.1 shows three possible scenarios that can occur in an adaptive proxy: a complete cache miss, a partial cache hit, and a complete cache hit. All three scenarios are simplified; a video consists only of two visual streams. In Figure 6.1 *SDP*– or *data*– means that an adapted version is sent to the client. We will explain a partial cache hit in detail.

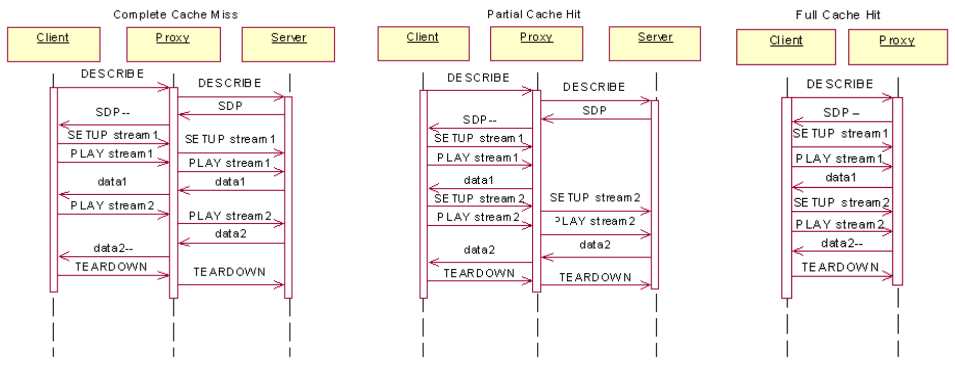


FIG. 6.1. *Caching scenarios*

Suppose that video-1 consists of 8 ESs (1 IOD, 1 OD stream, 1 BIFS, 1 audio stream, 2 video objects, each consisting of 2 ES). The proxy has already seen video-1 and has reduced each video object to its base layer.

- The client starts by sending its terminal capabilities to the proxy. For a first prototype, we have appended this information as an MPEG-21 *Digital Item* [14] directly to an RTSP DESCRIBE command. In the following example, a PDA with an 8-bit grey-scale display, a resolution of 240x320 and a network capability up to 128 kBit/sec, requests a video. The device supports mono audio.

```
DESCRIBE rtsp://127.0.0.1/videoandaudio.mp4 RTSP/1.0
CSeq: 1
Accept: application/sdp
User-Agent: MPEG4ITEC Player
Content-Type: application/mpeg21_dia
Content-Length: 458
<xml version="1.0" encoding="UTF-8"?>
<DIA xmlns="urn:mpeg:mpeg21:dia:schema:2003"
  xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mpeg:mpeg21:dia:schema:2003">
<Description xsi:type="UsageEnvironmentType">
  <Terminal>
    <InputOutput>
      <Display bitsPerPixel="8" colorCapable="false">
        <Resolution horizontal="240" vertical="320"/>
      </Display>
      <AudioOut numChannels="1"/>
    </InputOutput>
    <DeviceProperty>
      <Storage size="64.0" writable="false"/>
      <DeviceClass>PDA</DeviceClass>
    </DeviceProperty>
  </Terminal>
  <Network>
    <Capability maxCapacity="128000" minGuaranteed="32000"
           inSequenceDelivery="false" errorDelivery="true"/>
  </Network>
</Description>
</DIA>
```

- At the proxy, DESCRIBE triggers the creation of a currently empty *ServerSession* object associated with the RTSP port to the client. The *CacheManager* module is asked, if for this URL any ESs are cached. Based on this information and the terminal capabilities of the client, an SDP description is created and sent to the client. Additionally, the ServerSession object starts to create *DataChannels* (DCs); for each cached ES, one DC reading from a local file is created. If one ES is missing, the ServerSession object has to create a *ClientSession* object that has an RTSP connection to the media server and adds DCs, which read from the network, to the ClientSession object. Note that the ClientSession object immediately forwards the DESCRIBE command to the media server.
  If no MPEG-7 information is cached, the proxy fetches an MPEG-7 description from the media server. This description is used by cache replacement strategies in the proxy but could be used by a content provider to restrict or forbid adaptation. For instance, imagine a commercial video server environment where users pay a certain amount of money for watching a full-quality video; in this case the proxy should not modify the video in any manner.
  If the terminal capabilities indicate that the end-device cannot handle all ESs, some ESs are simply omitted from the SDP description. In the example, this is the enhancement ES from video object 2.
  If the terminal capabilities and the pre-cached MPEG-7 meta-data indicates that the client could handle an adapted version of the video, a *Filter/Adaptor* is set at the affected DCs.

- The client can now parse the SDP description and extract the desired number of ESs. An example for such a reply including IOD and one video ES description is:

```
RTSP/1.0 200 OK
Server: ItecMp4Server
CSeq: 1
Content-Type: application/sdp
Content-Length: 736
Date: 31 Jan 2003 11:52:22 GMT
Expires: 31 Jan 2003 11:52:22 GMT
Content-Base: rtsp://127.0.0.1/videoandaudio.mp4/

v=0
o=StreamingServer 1044010342 840 IN IP4 143.205.122.43
c=IN IP4 143.205.122.43
b=AS:112
t=0 0
a=control:*
a=mpeg4-iod:"data:application/mpeg4-iod;base64,AoCAgG0AT [..]"
a=isma-compliance:1,1.0,1
a=range:npt=0-13.23300
m=video 0 RTP/AVP 96
b=AS:112
a=rtpmap:96 MP4V-ES/90000
a=control:trackID=1
a=cliprect:0,0,240,180
```

```
a=fmtp:96 profile-level-id=1;config=000001000000012000044006685821 20A31F
a=mpeg4-esid:1
```

- For each ES, an RTSP SETUP request is generated:

  ```
  SETUP rtsp://127.0.0.1/videoandaudio.mp4/trackID=1 RTSP/1.0
  CSeq: 2
  Transport: RTP/UDP;unicast;client_port=40002-40003;
  ```

- The proxy sees 6 SETUP requests, which are forwarded to their corresponding DCs. Assuming that the OD, BIFS, and audio streams are cached, their DCs will generate immediate replies. The video objects are partly cached. Both ESs from video object 1 are requested, one of them is not cached. One ES is requested from video object 2, the second never showed up in the SDP and thus, is never requested. The ServerSession object simply forwards requests to non-cached streams to the ClientSession, which generates SETUP requests and forwards them to the media server.

- The client receives for each SETUP a response from the proxy:

  ```
  RTSP/1.0 200 OK
  CSeq: 2
  Server: ItecMp4Server
  Last-Modified: 31 Jan 2003 11:52:22 GMT
  Cache-Control: must-revalidate
  Session: 8162240
  Date: 31 Jan 2003 11:52:22 GMT
  Expires: 31 Jan 2003 11:52:22 GMT
  Transport: RTP/UDP;unicast;client_port=40002-40003;source=127.0.0.1;
  server_port=20000-20001
  ```

  It extracts the SessionId and the ports. Now the client has all the information necessary to create an RTP connection to the proxy and starts to send PLAY requests for the streams. The synchronization of the ESs is done by the client based on the timing information in the sync layer.

  ```
  PLAY rtsp://127.0.0.1/videoandaudio.mp4 RTSP/1.0
  CSeq: 8
  Session: 8162240
  Range: 0.0 - 13.233000
  ```

- The ServerSession object parses these requests and invokes the PLAY method at the corresponding DataChannel object.

- The session ends with a TEARDOWN request. The MP4Stream object is updated with the new data and inserted via the cache manager, resulting in a cache replacement (CR) algorithm to start. The CR algorithm chooses a video to adapt, it analyzes the adaptation possibilities extracted from the MPEG-7 document, and chooses from several adaptation sets the one that offers the best quality in relation to the resulting file size.

In the case of a complete cache miss, the scenario changes slightly because the meta-information is needed in advance:

- When the proxy receives the DESCRIBE request, it detects that nothing of the video is cached. A ServerSession object is created, which encapsulates a ClientSession object. The ClientSession object detects that it has to pre-fetch the OD stream and immediately sends a DESCRIBE, followed by a SETUP for the OD and the BIFS stream, the two streams which will always be requested by a client. Afterwards, the OD stream is parsed and an SDP description is generated and sent to the client. This pre-fetching increases the delay for the client, but the delay should be acceptable because both ESs are rather small.

7. Summary and Further Work. We have presented the design of an adaptive MPEG-4 proxy. We have shown how standards like MPEG-4, MPEG-7, and MPEG-21 can be used to realize a standard conforming, adaptive proxy. The use of MPEG-7 and MPEG-21 allows the components in a distributed multimedia system (media server, proxy and client) to communicate adaptation options in a standardized way; MPEG-4 provides the internal media structure to lower adaptation costs in the proxy. The modular design allows for different cache replacement strategies to be integrated and evaluated.

In addition, we support simple filter operations that work in the compressed and uncompressed domains that allow our proxy to act as a media gateway and to tailor a video to a client's terminal capabilities. While the implementation of the aforementioned filters is completed and first evaluations have been performed [11], the integration of system-level adaptation turned out to be a hard task. There are no free codecs available that allow one to create videos with native scalability support. As soon as tool and codec support is improving, we will integrate this adaptation possibility into our proxy. Another open issue is the side effect of the CPU intensive filtering operations on proxy behavior. If we want to maintain real-time behavior in the proxy, we are limited to a few adaptation steps in parallel. There are several possibilities to cope with this constraint. The first and simplest is to declare that the situation will improve with the availability of more advanced codecs. The second variant is to integrate optimized transcoding algorithms in our proxy that work in the compressed domain. Last but not least, extending the current single-node solution to a distributed proxy architecture is also an option.

Acknowledgments. This project was funded in part by FWF (Fonds zur Förderung der wissenschaftlichen Forschung) P14788 and by KWF (Kärntner Wirtschaftsförderungsfonds).

REFERENCES

[1] R. Koenen (ed.), *N4668 - Overview of the MPEG-4 Standard*, available at `http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm`, March 2002

[2] J. M. Martínez (ed.), *N4980 - MPEG-7 Overview*, available at `http://mpeg.telecomitalialab.com/standards/mpeg-7/mpeg-7.htm`, July 2002

[3] J. Bormans and K. Hill (eds.), *N5231 - MPEG-21 Overview*, available at `http://mpeg.telecomitalialab.com/standards/mpeg-21/mpeg-21.htm`, Oct. 2002

[4] H. Kosch, L. Böszörmenyi, and H. Hellwagner, *Modeling Quality Adaptation Capabilities of Audio-Visual Data*, In DEXA 2001 Workshop Proc., Sept. 2001

[5] S. Podlipnig, *Video Caching in Verteilten Multimedia-Systemen*, Ph.D. Thesis, University of Klagenfurt/ITEC, April 2002

[6] R. Rejaie and J. Kangasharju, *Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming*, In Proc. NOSSDAV, June 2001

[7] R. Rejaie, H. Yu, M. Handley, and D. Estrin, *Multimedia Proxy Caching Mechanisms for Quality Adaptive Streaming Applications in the Internet*, In Proc. IEEE Infocom'2000, Tel Aviv, Israel, March 2000

[8] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara, *Proxy Caching Mechanisms with Video Quality Adjustment*, In Proc. SPIE International Symposium on The Convergence of Information Technologies and Communications, August 2001

[9] M. Zink, J. Schmitt, and R. Steinmetz, *Retransmission Scheduling in Layered Video Caches*, In Proc. ICC 2002, New York, April 2002

[10] L. Böszörményi, H. Hellwagner, H. Kosch, M. Libsie, and P. Schojer, *Comprehensive Treatment of Adaptation in Distributed Multimedia Systems in the ADMITS Project*, In Proc. ACM Multimedia 2002 Conference, Juan Les Pins, France, Nov./Dec. 2002

[11] P. Schojer, L. Böszörményi, H. Hellwagner, B. Penz, and S. Podlipnig, *Architecture of a Quality Based Intelligent Proxy (QBIX) for MPEG-4 Videos*, In Proc. World Wide Web Conference 2003, Budapest, Hungary, May 2003

[12] A. Vetro, H. Sun, and Y. Wang, *Object-based transcoding for adaptable video content delivery*, In IEEE Trans. Circuits and Syst. for Video Technology, March 2001

[13] S. Podlipnig and L. Böszörmenyi, *Replacement Strategies for Quality Based Video Caching*, In Proc. IEEE Int'l. Conf. on Multimedia and Expo (ICME), Lausanne, Switzerland, Aug. 2002

[14] Moving Picture Experts Group, *ISO/IEC JTC 1/SC 29/WG 11 N5353: Text of ISO/IEC 21000-7 CD - Part 7: Digital Item Adaptation*, available at `http://www.itscj.ipsj.or.jp/sc29/open/29view/29n5204c.htm`, Dec. 2002

[15] M. Döller and H. Kosch, *Demonstration of an MPEG7 Multimedia Data Cartridge*, In Proc. ACM Multimedia 2002 Conference, Juan Les Pins, France, Nov./Dec. 2002

[16] M. Ohlenroth and H. Hellwagner, *A Protocol for Adaptation-Aware Multimedia Streaming*, In Proc. IEEE Int'l. Conf. on Multimedia and Expo (ICME), Baltimore, USA, July 2003

[17] R. Tusch, *Towards an Adaptive Distributed Multimedia Streaming Server Architecture Based on Service-oriented Components*, Joint Modular Languages Conference, Klagenfurt, Austria, August 2003

Edited by: Zsolt Nemeth, Dieter Kranzlmuller, Peter Kacsuk, Jens Volkert
Received: March 21, 2003
Accepted: June 5, 2003

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:
- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:
- programming environments,
- debugging tools,
- software libraries.

Performance:
- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:
- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:
- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in $\LaTeX 2_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the PDCP WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.