Scalable Computing: Practice and Experience
Volume 13, Number 4, pp. 339–350. http://www.scpe.org
ISSN 1895-1767
© 2012 SCPE

AN ALGORITHM FOR TRADING GRID RESOURCES IN A VIRTUAL MARKETPLACE

BENJAMIN AZIZ*

Abstract. This paper presents an algorithm for trading resources in Grids. Resource description includes main technical attributes of a resource, such as processing power, memory capacity, etc., as well as a price. Trading is performed in a marketplace where providers' resources are matched with consumers' demand by means of auction mechanisms. The matching algorithm follows a strategy where a consumer's demand is matched with providers meeting the technical requirement and the price closest to the one offered by the consumer.

Key words: Trading algorithms, Virtual Marketplace, Grid, Scheduling

1. Introduction.

All the advantages provided by utility computing, grid computing and virtualization do not only enable the execution of computationally intensive, scientific applications but also allow commercial customers to use the power of such a Grid to exectute their applications quickly, effectively and efficiently. However, there are many different kinds of users such as SMEs, large enterprises, academia, etc., distinguishing themselves in the amount of budget, urgency of their application, and quality of service expectations. For example, users who require the termination of the execution of their application within specific period of time, are willing to pay a higher price than other users. There exists a wide variety of related market systems which are based on fixed prices, bartering, negotiations or auction models for leasing Grid contracts. These systems include GridEcon [1], SORMA [15], BREIN [7], BEinGRID [19], Edutain@Grid [6] and GRIA [21]. However, very few efforts have been made to fully specify the design of a market that is tailored for the Grid resources and services.

A Virtual Marketplace of Resources (VMR) is a marketplace, which comprises all the functionality for leasing of computational services for a time period so as to use Grid resources effectively and efficiently. A VMR allocates Grid resources according to their specifications to applications as a means of meeting performance goals described by the application provider to the available resources. The VMR system architecture we consider here was developed earlier in the context of project XtreemOS [23].

The main contribution of this paper is to introduce a marketplace trading algorithm that facilitates the commercialisation of Grid resources, where a provider is capable of listing the Grid resources, and buyers demand the required computing resources for their applications on the basis of utility (e.g. price/number of resources/time). The VMR algorithm offers a public Internet market that would be open to registered users to buy and sell computing resources. The ability to utilize remote Grid computing platforms frees both the provider and the buyer from the need to own or acquire the necessary computational infrastructure. Furthemore, the marketplace system will provide an infrastructure that will allow end-users not only to consume but also to sell services and resources on the Grid. Therefore, creating a new economy in which all users can actively participate. VMR offers a solution to both the high cost of ownership and the fluctuating usage patterns of computing capacity.

The VMR system facilitates the creation of self-governing collections of providers and buyers that make resource allocation decisions strictly based on current price/resource availability. Providers and buyers act autonomously to improve their own standing in a market. The price and resource specifications mentioned by the provider/buyer are their own choice and can adapt any strategy/technique as conditions change. Buyers usually want to pay the least amount possible for the resources needed to execute their application. Providers, on the other hand, wish to generate greater and greater revenue and larger profits from their offered resources.

VRM considers a fixed price approach instead of performance management approaches based on Service Level Agreements (SLAs) and utilization. Once the match has been allocated between a resource and and application, the published price has to be paid by the buyer to the provider after the execution of the application. Once the payment has been made the result of the execution is sent to the buyer.

This paper is organised as follows: Section 2 presents the computing resource exchange related work. In Section 3, we give an abstract view of the VMR solution. Section 4 mentions how the demands and offers are be described and when the trading algorithm is being activated. Section 5 describes the trading algorithm,

*School of Computing, University of Portsmouth, Portsmouth PO1 3HE, United Kingdom(benjamin.aziz@port.ac.uk).

and Section 6 explains how the trading algorithm is being executed with the help of an example, and finally, Section 7 concludes the paper giving directions for future work.

2. Related work. Several research systems [3] have explored the use of different economic models for trading resources to manage resources in different application domains: CPU cycles, storage space, database query processing, and distributed computing. Despite the fact that there are a few commercial providers of utility computing (e.g. Amazon [18], HP [10], IBM [9], Google [8], Sun [20]), these providers (being commercial) offer their resources at a relatively expensive cost. If compute resource users (providers and buyers) accept and trust the computing resource exchange for executing their trades, it will increase the supply of computing resources in the market. Consequently, computing resources price will decrease and become affordable to a low budget enterprises. Two open source systems SORMA [14] and GridEcon [1] have been developed in order to attract customers to computing resource exchanges.

SORMA [15] uses self-organizing resource management system to develop methods and tools for an efficient market-based allocation of resources. SORMA provides a flexible market infrastructure, which can access resources over different virtualisation platforms and enable different resource managers to plug in the market. SORMA follows the bottom up strategy, means it first define the market mechanisms for trading the resources and then design the appropriate middleware components for brokering, accounting, and charging. SORMA is being developed to offer the possibility of loosely integrating emerging Grid markets.

On the other hand, GridEcon [2] provides computing resource exchange for commoditised computing resoruces by offering a set of services that could help new users to accept the computing resource exchange concept. GridEcon is a computational resource auctioning system built upon a bid matching algorithm. Offers submitted by the resoruce buyers and providers are matched to execute the application on the cheapest available resource. GridEcon follows the top down strategy, means it first identify the kind of higher-level goods that applications would like to obtain in a commercial Grid environment and then defines the appropriate business model.

While SORMA focuses on the openness of decentralised complex service markets and GridEcon addresses an exchange for basic Computing resources, VMR is a compliment to these systems as VMR is an auctioning system that provides decentralised computing resource exchange to access resources over different platforms. VMR is independent of any underlying Grid middleware of the platform. VMR matches demands and offers and executes them against each other only after verifying the terms and conditions defined by the users.

Other more recent works have also addressed the problem of establishing a marketplace of Grid/Cloud resources. For example, in the Polish Agents in Grid (AiG) project [22], software agents are used to provide a meta-level Grid middleware where economic models can be established based on Service Level Agreement (SLA) representations of the Grid resources and clients. In this middleware, owners can make their resources available and clients can commission those resources for the execution of jobs, after SLAs of both sides are negotiated.

In [24], the authors propose also a multi-agent system for carrying out automated negotiations in any service-oriented environment including Grids. Similarly, in [11], the authors use the cost of electrical power as the unit of cost in a model of scheduling they propose for environments of distributed servers. This approach provides a more concrete realisation of the concept of cost than in our case and the case of other models.

Business-oriented large-scale systems, such as Clouds, have also adopted SLA-based trades. These include for example OPTIMIS [16], mOSAIC [13] and Cloud4SOA [4]. In this paper however, we stay within the scope of marketplace research carried out in the context of Grid system.

3. The Concept of a Virtual Market Place. At its heart, a VMR facilitates the commercialization of Grid resources on-demand through a virtual marketplace of computational resources, where a seller is capable of listing the Grid resources, and buyers can ask/bid dynamically for required computing resources for their applications. VMRs assume that resources are available in Grids based on various technologies (e.g. Globus-based, gLite-based Grids etc.). One such VMR was developed within the scope of project XtreemOS [5, 12]. XtreemOS aimed at building a Grid-based distributed operating system that provided a single abstraction of physical hardware and software services offered by a collection of standalone Linux operating systems to users within a Grid.

Such VMR system aims at providing a computational resources auctioning system built upon a dynamic bid matching algorithm tailored specifically for the trading of computing power. It helps both consumers and providers of computational resources to use the resources efficiently so as to maximize the economical benefits and minimize the idle time for them. The XtreemOS VMR was developed to integrate into a single framework three key features:

- *Interoperability* is achieved by using a standard programmable interface, the Simple API for Grid Application (SAGA) [17], to bridge the gap between existing Grid middleware and application level needs. The same system could run on any Grid platform (e.g. XtreemOS, gLite etc.), or interoperate on Grids using resources from other platforms.
- *Cost saving for end users* is guaranteed by allocating the applications to the more economical resource(s), following policies defined by the end users.
- *Dynamic scheduling* is achieved through the virtual marketplace, which implements scheduling and trading algorithms that allocates applications following classical performance parameter as well as the cost of resource usage.

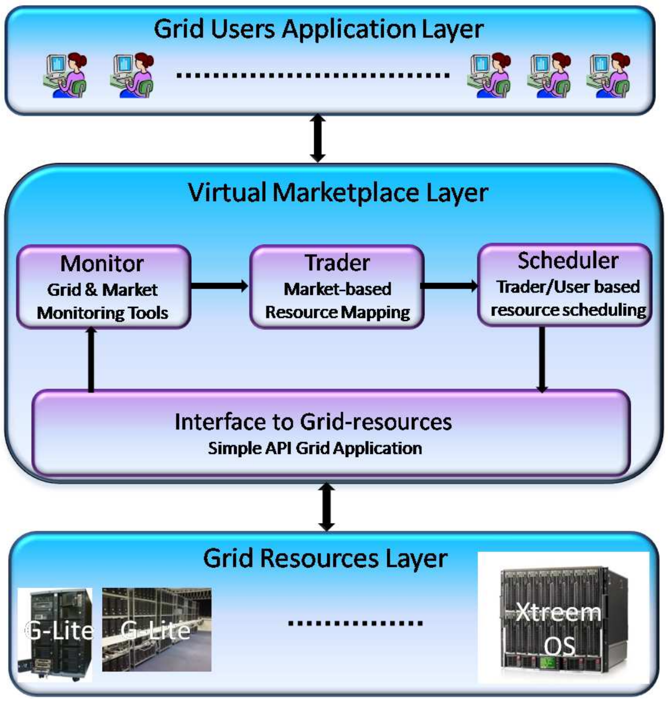
Grid Users Application Layer
Virtual Marketplace Layer
Monitor
Grid & Market Monitoring Tools
Trader
Market-based Resource Mapping
Scheduler
Trader/User based resource scheduling
Interface to Grid-resources
Simple API Grid Application
Grid Resources Layer
G-Lite
G-Lite
Xtreem OS

Fig. 3.1: Abstract view of VMR [23].

Figure 3.1 (originally from [23]) illustrates a logical layered architecture of the virtual marketplace defined in XtreemOS. This architecture represents the entities and their dependency to other entities, where the flow of information or control is depicted by arrows. An arrow from an entity X to an entity Y means that X sends information to Y or passes control to Y. Our system offers the mechanisms for deploying and executing the application (e.g. automatic deployment, execution monitoring, and hardware resource discovery) for the business processes to purchase the resources on the Grid. Implementing such a business model requires at least the following basic roles, which belong to three layers: the Grid users application layer, the Virtual marketplace layer, and the Grid resources layer.

Grid users application layer: This layer allows end-users (scientist, chemist, physician etc.) to submit applications to the deployed resources. We consider Grid application to be a collection of work items to solve a certain problem or to achieve desired results using the Grid infrastructure. Grid applications can be scientific, mathematical, academic problems or the simulation of business scenarios, like stock market development, that require a large amount of data as well as a high demand for computing resources in order to calculate and handle the large number of variables and their effects.

Virtual marketplace layer: This layer implements monitoring, trading and scheduling services so as to utilize the available Grid resources efficiently and exploit the benefits of the interoperability and scalability of the Grid platform. This layer consists of four main components:

- *Monitor* implements the monitoring/reporting techniques, which monitors resources and reports changes such as dynamic re-allocation of resources, according to changes generated from evolution in the resource market,execution status of submitted applications, etc. Monitoring is achieved by either direct or indirect capture of resource status and pre-defined events. The indirect interface uses logs generated at run-time by the Grid infrastructure. The direct interface is a portal collecting dynamically events generated by monitoring services associated to the Grid infrastructure.

- *Trader* implements the trading algorithms that depends on criteria such as cost, processing power, execution time or resource availability. It is also responsible for sending notifications to users about the status of their request. For example, inform a bidder whether the bid is winning or not.
- *Scheduler* schedule the application on to the selected Grid resource. Scheduling of the end users application is done on to the selected resource by following the analysis provided by the trader.
- *Interface to Grid resources* provides simple access for distributed systems and abstractions for applications and thereby address the fundamental application design objectives of interoperability across different infrastructure. It also supports job submission and data management (efficient data access, data replication, streaming of data, etc.).

Grid resources layer: This layer consist of the resources (server, storage and network) used to execute the end users applications. The submitted application is executed on the selected Grid resources and result is sent back.

4. VMR Marketplace Mechanism and Matching module. In our marketplace, Buyers and providers interact through the marketplace by means of the broker, in order to lease/offer Grid resources. Providers are indifferent regarding how their machines will be consumed in the market, i.e. what kind of needs the consumers want to accommodate by leasing the providers' resources. The providers solely offer their resources and it is the responsibility of the marketplace to match them with the consumers' demand by means of a market mechanism and a matching algorithm.

A buyer's order is specified by means of the total number of resources (with its required specification) that must be made available up to a specific time interval, so that a certain computationally-intensive task is executed in time. All parties publicly announce the maximum price they are willing to buy for and the minimum price they are willing to sell for. All buyers should mention the maximum price they are willing to buy for and all the providers should mention the minimum price they are willing to sell for the leasing of resources in a specified time interval. These prices, resource information, participants' information are recorded and put in a database.

Below we define the demand (resp. the offer) that the buyers (resp. the providers) submit to the VMR in order to express their services, which comprises of computational elements that are made available in a time interval whose start and end time are specified upon submission and are not flexible.

A demand describes the buyer's requirements. Demand is specified as:

- (1) R_b - The resource specification (processing power, hard disk, RAM, OS, etc.),
- (2) Q_b - The number of resources that are demanded,
- (3) S_b - The start time of the interval for using the resources,
- (4) D_b - The time duration, for which resources are needed,
- (5) P_b - The price expressed in £for use of one resource/min, and
- (6) Et_b - The expiration time of the demand.

An offer describes the resources posted by the providers. Offer is specified as:

- (1) R_p - The resource specification (processing power, hard disk, RAM, OS, etc.),
- (2) Q_p - The number of resources that are available,
- (3) S_pStp - The start time of the interval when the resources are available,
- (4) D_p - The time duration, for which resources are available,
- (5) P_p - The price expressed in £for use of one resource/min, and
- (6) Et_p - The expiration time of the offer.

When more than one offer/demand is added to the queue at same time than sort according to submission expiry time if two same then sort according to the start time.

Buyers/providers orders will not be immediately fulfilled unless there is a previously posted compatible reciprocal demand/offer. All the compatible trades i.e., when the buyers demand price exceeds the providers offer price for a match between an application requirements and resoruce specification, are immediately executed. If no compatible reciprocal offers/demands are available, the offer/demands remain in the respective queue until they are matched in the future or expire.

The matching module is invoked whenever a demand or an offer is submitted to the VMR. The rationale behind the matching module is summarized as follows:

- a) Demands are completely satisfied, i.e. there are never remainder demands, pieces of the same demand that are still pending. This is not the case for offers, which can be partly matched in order to serve demands.
- b) Each demand is served by one provider.
- c) Matching solution ensures that the demanded resources are allocated throughout the service time interval (application's demanded duration), so that the resources switching is avoided over time.

Matching module activates in the following two events:

- 1) A new demand is submitted by the buyer: Matches candidates for a demand (whose price is P_b) only those offers (whose price P_p) where $P_b \geq P_p$ holds. Therefore, we omit examining higher price offers and try combining them with lower price offers, even if such combinations could in fact serve the demand.
- 2) A new offer is submitted by the provider: Matches candidates for an offer (whose price is P_p) only those demands (whose price P_b) where $P_b \geq P_p$ holds. Therefore, we omit examining lower price demands and try combining them with higher price demands.

The rationale of the matching procedure is to provide the required coverage of a) the demand with the cheapest matching offer and/or b) the offer with the equal or higher matching demand by means of a matching algorithm. If a demand is matched fully then reservation of resources, accounting and computation of remainder offer that replace the original offer in the offer queue are performed; and the demand is removed from the demand queue and subsequently serviced. On the other hand, if an offer is matched fully then reservation of resource and accounting are performed; and the offer is removed from the offer queue and the demand is removed from the demand queue, which is subsequently serviced.

As a demand is always fully matched, this is not the case for an offer. Therefore, in general a fraction of an offer may be used to (partly) match and serve a demand, thus generating remainder offers. Thus, when an offer is matched partially, the reservation of resources, accounting and computation of remainder offer that replaces the original offer in the offer queue is performed; and the matched demand is removed from the demand queue and subsequently serviced.

It is the responsibility of the matching module to be invoked periodically, in order to compute matches and remove expired offers and demands from the offer/demand queue. The results of the matching procedure are subsequently passed to the scheduler and the accounting system of the market place.

5. Trading algorithm. The algorithm defines how demands (submitted by the buyers) and offers (submitted by the providers) are matched. VMR trading algorithm is based on market mode because it offers a control strategy that is computationally efficient, flexible in the face of emergent behavior, and makes visible to IT personnel mission-critical price-performance statistics that directly reflect the marketplace's ability to deliver infrastructure tailored to real business value. The market-model trader used in VMR is capable of trading any kind of resources (compute resources, storage resources etc.), as long as the resource's consumption requirements can be translated into the trader's key-value format.

Currently, we assume that demands should be fully served by resources of a single provider, however in the future, buyers may be allocated resources of multiple providers, as long as each of these is reserved for the buyers throughout the specified time interval. This assumption is imposed due to technological constraints, since there may be significant switching costs when shifting unfinished computing jobs between virtual machines of different providers within the service time interval.

Psuedo codes 5.0.1 and 5.0.2 next present the VMR trading logic that is executed when a new demand/offer is submitted. As a first prototype, a meaningful matching procedure for the VMR is to try matching a demand with the cheapest matching offers. In future other matching procedures can be plugged-in to do matching according to the company preferences/constraints such as buyer can specify the list of providers they would like to submit their application for execution.

The trading algorithm runs from scratch, whenever a new demand arrives or a new offer arrives to perform a matching between the offers and demands respectively. When an offer arrives that match a demand, three things have to be decided a) how much of it to use, b) where to place it and c) what to do if offer is not completely used. The solution we adopt is a) order the matching offers according to the demand's time constraints i.e., use till demand is completely fulfilled, b) place it meet the demand's deadline such that offer is divided in minimum

Algorithm 5.0.1 when a new demand is submitted by a buyer.

```
if $offer\_queue \neq null$ then
  select the offer where $R_b == R_p$
  store the selected offers in ascending order of offer prices $P_p$
  Select the offers where $P_b \geq P_p$
  if selected_offer_queue≠null then
    $i \leftarrow 0$
    while i < sizeof(selected_offer_queue[]) do
      if $((Q_p[i] \geq Q_b)$ &&$(D_p[i] \geq D_b))$ then
        begin_time= max$(St_p[i], St_b)$
        end_time=min$(Et_p[i], Et_b)$
        if (end_time-begin_time) $\geq D_b$ then
          if begin_time == $St_p[i]$ then
            Computation start time $S_t$=begin_time
          else
            Computation start time $S_t$=end_time-$D_b$
          end if
        end if
        pass information (selected_offer_queue[i], $S_t$, $D_b$)to the scheduler
        if $(D_p > D_b[i])||(Q_p > Q_b[i])$ then
          calculate the remaining offer using Pseudo code 5.0.3.
          resubmit the new created offers
        else
          remove the offer from the offer queue
        end if
      else
        i++
      end if
    end while
  else
    put the demand in the demand queue
  end if
else
  put the demand in the demand queue
end if
```

chunks and c) if offer left with services/time to be used, remaining offer is calculated according to pseudo code 5.0.3 and resubmitted.

Trading is performed by means of an auction mechanism. The submitted demands and offers are placed in the demand queue and the offer queue respectively. If two or more orders at the same price appear in an allocated queue, then they are entered by time with older orders placed above the newer orders. Trader collects orders from buyers and providers and executes trades (makes a call) periodically to clear the market by matching buyers with providers. A demand/offer remains in the queue until it is allocated, removed due to its expiration time or removed by the submitted user. Resources are allocated for a specified amount of time that is required by the application and defined by the buyer.

The trading algorithm initially computes the candidate matches to demand by means of creating a matrix as shown in Figure 5.1. Each column of the matrix corresponds to a time slot (i.e. the time interval in which service can be provided). Each row corresponds to a provider that can offer service now, with the cheapest being on the top row. A cell of the matrix is marked if the provider can offer computing resources during this specific time slot, as shown in Figure 5.2.

Complexity of buyers and providers trading algorithm: Counting the total number of basic operations, those which take a constant amount of time in Psuedo code 5.0.1 followed by the while loop where the value of i changes every time through the loop (N + (N - 1) + · + 2 + 1 = N(N+1)/2), the total number

Algorithm 5.0.2 when a new offer is submitted by the provider.

```
if $demand\_queue \neq null$ then
  select the demands where $R_b == R_p$
  store the selected demands in descending order of offer prices $P_p$
  Select the demands where $P_b \geq P_p$
  $i \leftarrow 0 j \leftarrow 0$
  while selected_demand_queue≠null do
    if $((Q_p \geq Q_b[i])$ &&$(D_p \geq D_b[i]))$ then
      begin_time= max($St_p, St_b[i]$)
      end_time=min($Et_p, Et_b[i]$)
      if (end_time-begin_time) $\geq D_b[i]$ then
        selected_demands[j]=selected_demand_queue[i]
        calculate sorting_condition[j] = $D_b[i] * Q_b[i] * P_b[i]$
        j++
      else
        i++
      end if
    end if
  end while
  if selected_demands≠null then
    Sort selected_demands in descending order of sorting_condition.
    Select the selected_demands[0]
    if $St_{selected\_demands[0]} > St_p$ then
      Computation start time $S_t$=$St_{selected\_demands[0]}$
    else
      Computation start time
      $S_t$=min($Et_p, Et_{selected\_demands[0]}$)-$D_{selected\_demands[0]}$
    end if
    pass information (selected_offer_queue[i], $S_t$, $D_b$)to the scheduler
    Remove the demand from demand queue
  end if
  if ($D_p > D_b$[selected_demands[0]])||( $Q_p > Q_b$[selected_demands[0]]) then
    calculate the remaining offer using Psuedo Code 5.0.3
    resubmit the new created offers
  end if
else
  put offer in offer queue
end if
```

of operations is equivalent to $O(N^2)$. As the runtime complexity of Psuedo code 5.0.2 is less than Psuedo code 5.0.1, we can say that Psuedo code 5.0.2 is more efficient than Psuedo code 5.0.1. However, both falls into $O(N^2)$ complexity class.

6. Implementation of Virtual Marketplace of Resoruces. To explain the trading algorithm, we assume that VMR has a demand queue as shown in Figure 6.1 and an offer queue as shown in Figure 6.2 with respect to the matching martix of Figure 5.1. For simplicity, the time in demand/offer queue is considered to be of same day and even the resource specifications is limited to the OS only. However, VMR uses the timestamp datatype for describing the time which enables the provider/buyer to specify the time from future dates. Similarly resoruce specification is not only limited to the OS, a provider/buyer can speciy the resoruce's hardware and software description along with the versions.

First case is when a new offer is submitted by a provider. For example, Provider P offers 8 XtreemOS resources for 5 hrs, starting at time 11:00, with offer price £0.03, and time limit 23:30. Following the Psuedo Code 5.0.2 buyers B_1, B_3, B_5 are selected and sorted in descending order of price from the demand queue, i.e., B_5, B_3, B_1. After comparing the price $P_{B_5}, P_{B_3}, P_{B_1} \geq P_p$ only demands from B_5 and B_3 are added to the

Algorithm 5.0.3 calculate the remainging offer.

```
if $S_t == St_p$ then
  if $Q_b < Q_p$ then
    Submit new offers for partial used resources as $(S_t + D_b, Et_p, P_p, D = D_p - D_b, Q_b)$ and totally unused resoruces as $(St_p, Et_p, P_p, D = D_p, Q = Q_p - Q_b)$
    (Case of Fig. 5.2(1))
  else
    Submit new offer as $(S_t + D_b, Et_p, P_p, D = D_p - D_b, Q_p)$
    (Case of Fig. 5.2(2))
  end if
else
  if $S_t + D_b == Et_p$ then
    if $Q_b < Q_p$ then
      Submit new offers for partial used resources as $(St_p, Et_p - D_b, P_p, D = D_p - D_b, Q_b)$ and for unused resoruces as $(St_p, Et_p, P_p, D = D_p, Q = Q_p - Q_b)$
      (Case of Fig. 5.2(3))
    else
      Submit new offer as $(St_p, Et_p - D_b, P_p, D = D_p - D_b, Q_p)$
      (Case of Fig. 5.2(4))
    end if
  else
    if $Q_b < Q_p$ then
      Submit new offers for partial used resources before use as $(St_p, St, P_p, D = D_p - D_b, Q_b)$, after use as $(S_t + D_b, Et_p, P_p, D = D_p - D_b, Q_b)$, and unused resoruces as $(St_p, Et_p, P_p, D = D_p, Q = Q_p - Q_b)$
      (Case of Fig. 5.2(5))
    else
      Submit new offers for only partial used resources before use as $(St_p, St, P_p, D = D_p - D_b, Q_p)$ and after use as $(St + D_b, Et_p, P_p, D = D_p - D_b, Q_p)$
      (Case of Fig. 5.2(6))
    end if
  end if
end if
```

selected_demand_queue. First quantity and duration required by the buyer B_5 is checked. Possible available duration is calculated by matching the start time and expiry time of buyer B_5 and provider P, which is similar to the Figure 6.3(6). As available duration is more than the required duration by B_5, the demand is added to the slected_demands queue and its sorting_condition is calculated (3*3*0.06) as 0.54. Then next demand from the selected_demand_queue is selected i.e., demand from buyer B_3 and same procedure the repeated as mentioned for the demand by buyer B_5. Now the selected demands B_5 and B_3 are sorted in descending order of the sorting condition, which conclude B_3's demand to be the matching demand for the P's offer. Time slot for the matching matrix is calculated (shown in Figure 5.1). Demand is removed from the demand queue.As provider P has offered quantity and duration is more than used by the demand of B_3, remaining offer is calculated using Pseudo code 5.0.3. As starting time S_t is calculated to be 19:00 which is neither St_p (11:00) nor Et_p (23:30), however Q_{B_3} (5) is less than the Q_P (8), the remaining offers (Figure 5.2(6)) are resubmitted to the VMR.

Second case is when a new demand is submitted by a buyer. For example, Buyer B demands for 3 Linux resources to be used for 2 hrs, starting at time 13:00, with demand price £0.05/resource/min, and time limit 23:00. Following the Pseudo Code 5.0.1, providers P_2 and P_5 offers the similar resources as required by the buyer B. P_2 and P_5 are sorted in ascedning order of price and placed in selected_offer_queue as both has $(P_{P_2}, P_{P_5}) \geq P_B$. First, the quantity and duration offered by P_5 is checked and possible available duration is calculated (which is similar to Figure 6.3(7)). As available duration is greater than the demanded duration, the match between P_5 and B is added to the matching matrix. Provider P_5 has offered more quantity and duration than used by the demand, remaining offers are calculated using the Pseudo code 5.0.3. As end time of the execution is equal to Et_{p_5}, the remaining offers (Figure 5.2(1)) are resubmitted to the VMR.

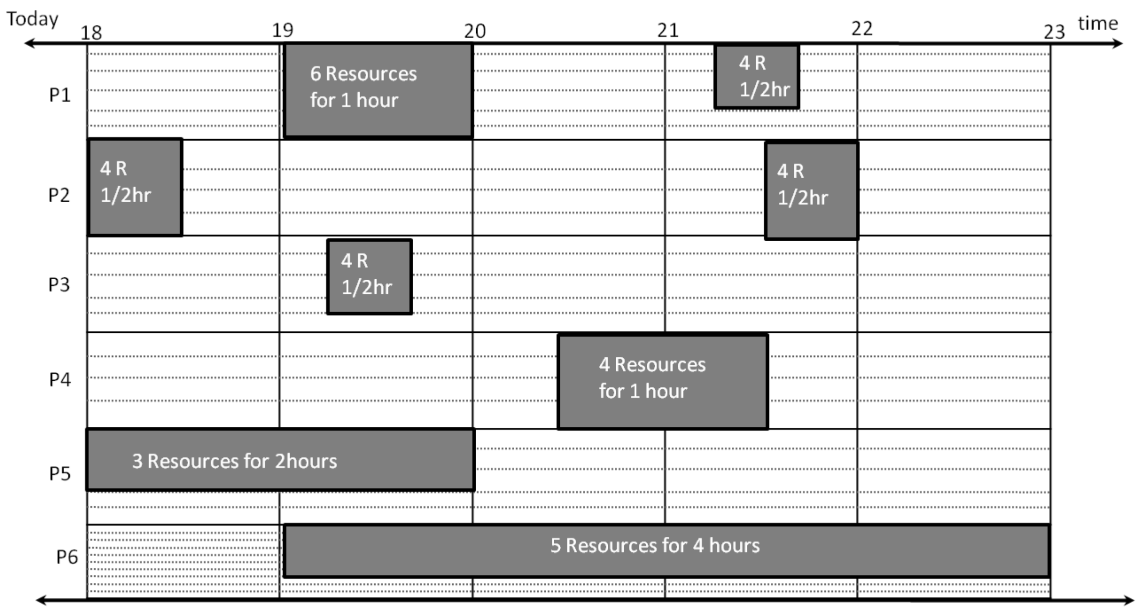

Fig. 5.1: Matching Matrix.

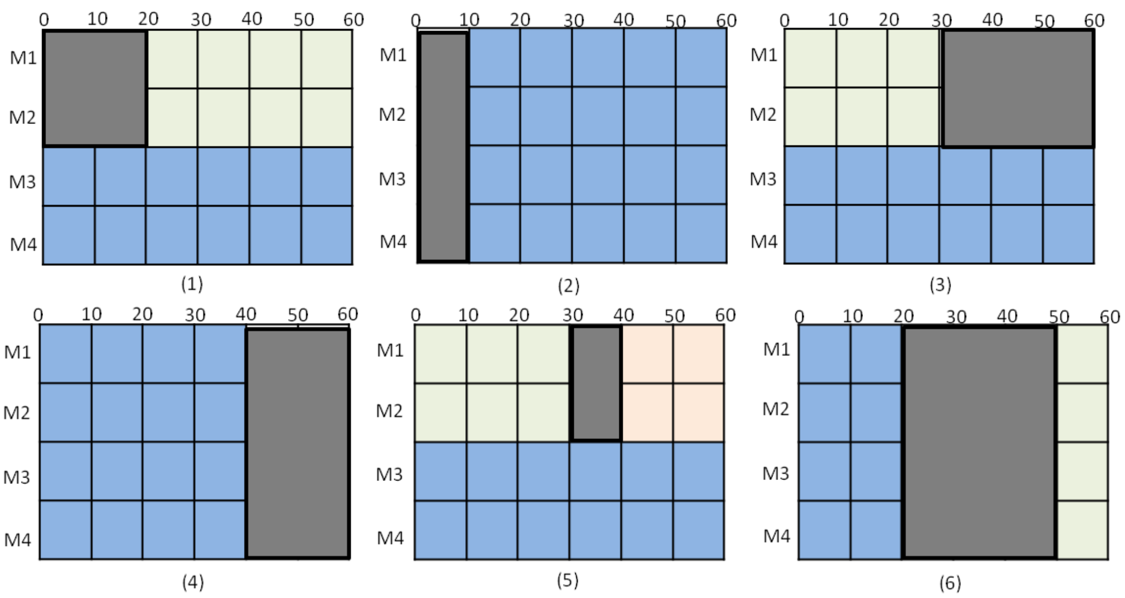

Fig. 5.2: Grey: Matching (demand and offer) possibilities, other colors: recalculated offers.

VMR GUI is a visualisation tool that provides an online marketplace for provider and buyers to publish their products and needs. Matches that have been made by the trading algorithm are also displayed.

The VMR GUI provides a portal for new users to register and check the current status of the market. Currently, the GUI displays the demands/ offers of buyers/providers and matches that has been made by the trading algorithm. GUI Development Language is English and database is MySQL. Resources from two different Grids (gLite and XtreemOS) are added to the database. To achieve interoperability when using resources from different platforms SAGA [17] is considered.

Buyers	Requirement	Qunatity	Duration	Price	Start time	Expiry time
B_1	XtreemOs	4	3	0.02	12:00	20:00
B_2	gLite	9	3	0.04	10:00	18:00
B_3	XtreemOs	5	4	0.03	15:00	23:00
B_4	Windows XP	3	2	0.05	18:00	23:00
B_5	XtreemOs	3	3	0.06	10:00	20:00

Fig. 6.1: A demand queue.

Providers	Requirement	Qunatity	Duration	Price	Start time	Expiry time
P_1	gLite	5	3	0.04	8:00	16:00
P_2	Linux	4	4	0.05	10:00	20:00
P_3	Windows XP	2	5	0.04	12:00	23:00
P_4	gLite	10	5	0.03	9:00	18:00
P_5	Linux	5	3	0.04	18:00	23:00

Fig. 6.2: An offer queue.

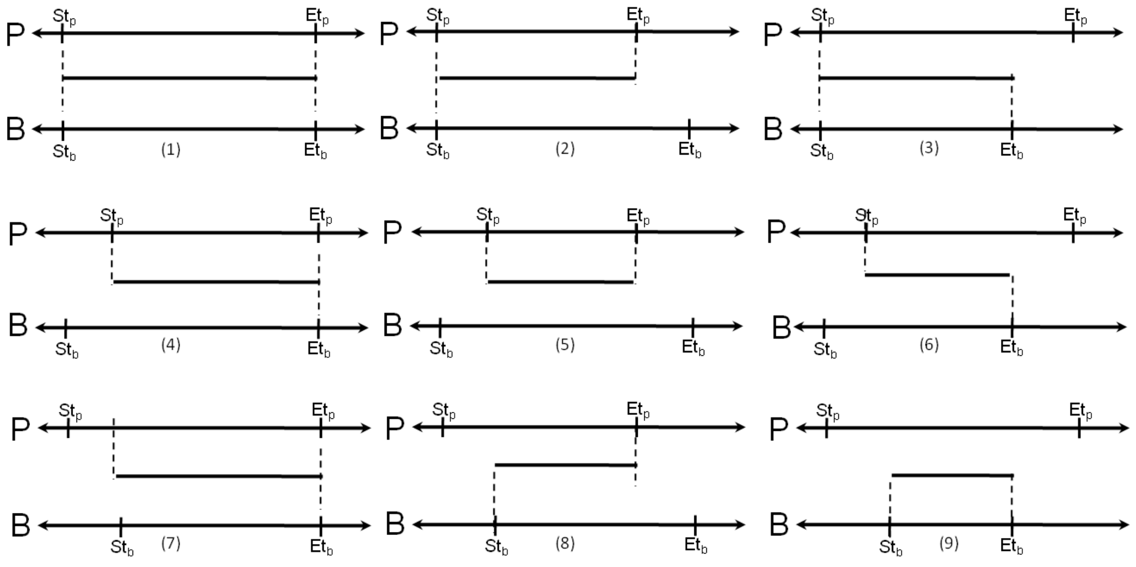

Fig. 6.3: Possibilities of demand and offer start and expiry time combinations.

Each resource allocation requires:

- *Resource specification*: This specification contains the application's requirement list. The list is used to specify which operating system, how much memory, which software, etc. are needed. Time Frame specification: This specification defines a window of time (in seconds) within which the allocation is to be considered valid. Applications execution start time, duration and deadline time should be specified by the buyer and similarly, the resource availability start time, duration and deadline time should be specified by the providers.
- *Periodic matching*: This time in seconds/minutes defines the interval at which VMR will execute the trading algorithm. A periodic cycle consists of the steps followed to process and execute trades. At the end of the trading cycle, compatible demands and offers are paired up. Demands and offers that are not converted into successful trades at the end of the trading cycle will wait in the queue till the next trading cycle, until cancelled or matched. The length of the cycle depends upon the type (CPU,

storage) and purpose or sector (a user community like research, bank) or can be fixed as in our first prototype to be 120seconds.

- *VMR Catalog*: It is an organized and searchable repository of resources, providers and buyers information. The information is composed of a variable length amount of metadata in the form of name-value pairs that describes performance requirements or capabilities. Each entry in the catalog represents a single entity (resource, provider, and buyer) and contains metadata that describes the entities' attributes, properties, performance requirements and functionality. The catalog can be searched by specifying a query composed of a set of metadata that must match the metadata of one or more entries to be included in the result set.

7. Conclusion. This paper presents an algorithm for trading resources in Grids. Resource description includes main technical attributes of a resource, such as processing power, memory capacity, etc., as well as a price. Trading is performed in a marketplace where providers resources are matched with consumers demand. The VMR solution presented in this paper answers questions such as "which Grid resource should be used that will minimize cost along with achieving efficient applications' execution time?", "how end-user can select Grid resources according to pre-defined policies, including cost policies?" and "how to achieve interoperability when using resources from different platforms?".

For future work, we would like to extend the algorithms to allow them to become more flexible incorporating other criteria that may affect the decision for selecting offers and demands. The presence of several factors, not just cost, would imply a trade-off-based algorithm, which will be able to provide compromises among the different criteria. For example, one such criterion could be matching the security requirements (privacy, assurance, or risk-based) as expressed by the policies of the buyers and providers. Usually, security comes at an increased cost. So, depending on the level of assurance provided by the provider, the buyer may be willing to pay more for better security.

The current form of the algorithm assumes that the buyer or the provider is a single entity or organisation. In the future, we would like also to consider federations of entities (buyers/providers) or virtual organisations. This would have implications on the underlying architecture, as it would imply some form of synchronisation or consensus among the different entities comprising the federation as to how the price of the offer or demand is reached.

Currently trading algorithm consider that each demand is served by one provider, it will be useful if more provides together can serve the demand. For example, if a demand of 10 resources arrives and no one provider can serve the demand, however, 3 providers together can do so than demand should be matched, instead of adding to the demand queue. Even, once the match has been made, the application is executed on the selected resoruces even if any cheaper resoruces become available during the course. On the fly switching of application to cheaper suitable resource is also under consideration.

REFERENCES

[1] J. Altmann, C. Courcoubetis, J. Darlington, and J. Cohen, *Gridecon - the economic-enhanced next-generation internet*, in GECON'07: Proceedings of the 4th international conference on Grid economics and business models, Berlin, Heidelberg, 2007, Springer-Verlag, pp. 188–193.

[2] J. Altmann and S. Routzounis, *Economic modeling of grid services*, 2006.

[3] R. Buyya and S. Venugopal, *MARKET-ORIENTED COMPUTING AND GLOBAL GRIDS: An Introduction*, in Market-oriented Grid and Utility Computing, John Wiley and Sons, Hoboken, NJ, USA, 2010, ch. 1, pp. 3–27.

[4] Cloud4SOA, *Cloud4soa*. World Wide Web electronic publication. *http : //www.cloud4soa.eu/* accessed 04-Jan-2013.

[5] T. Cortes, C. Franke, Y. Jégou, T. Kielmann, D. Laforenza, B. Matthews, C. Morin, L. P. Prieto, and A. Reinefeld, *XtreemOS: a Vision for a Grid Operating System.* XtreemOS Technical Report # 4, May 2008.

[6] T. Fahringer, C. Anthes, A. Arragon, A. Lipaj, J. Müller-Iden, C. Rawlings, R. Prodan, and M. Surridge, *The edutain@grid project*, in 4th International Workshop on Grid Economics and Business Models, D. J. Veit and J. Altmann, eds., vol. 4685 of LNCS, Springer, August 2007, pp. 182–187.

[7] H. M. Frutos and I. Kotsiopoulos, *Brein: Business objective driven reliable and intelligent grids for real business*, International Journal of Interoperability in Business Information Systems, Issue3 (1) (2009).

[8] Google, *Run your web apps on google's infrastructure.* World Wide Web electronic publication. *http : //code.google.com/appengine* accessed 07-Oct-2010.

[9] IBM, *e-business on demand: A developer's roadmap.* World Wide Web electronic publication. *http : //www.ibm.com/developerworks/ibm/library/ i − ebodov/index.html* accessed 07-Oct-2010.

[10] H. Labs, *Utility computing services.* World Wide Web electronic publication. *http : //www.hpl.hp.com/research/about/ utility_services.html* accessed 07-Oct-2010.

[11] S. Mani and S. Rao, *Operating cost aware scheduling model for distributed servers based on global power pricing policies*, in Proceedings of the Fourth Annual ACM Bangalore Conference, COMPUTE '11, New York, NY, USA, 2011, ACM, pp. 12:1–12:8.

[12] C. Morin, *XtreemOS: A Grid Operating System Making your Computer Ready for Participating in Virtual Organizations*, in Proceedings of the Tenth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2007), IEEE Computer Society, 2007, pp. 393–402.

[13] mOSAIC, *mosaic cloud*. World Wide Web electronic publication. $http : //www.mosaic - cloud.eu/$ accessed 04-Jan-2013.

[14] D. Neumann, J. Stoesser, A. Anandasivam, and N. Borissov, *Sorma - building an open grid market for grid resource allocation*, in GECON'07: Proceedings of the 4th international conference on Grid economics and business models, Berlin, Heidelberg, 2007, Springer-Verlag, pp. 194–200.

[15] J. Nimis, A. Anandasivam, N. Borissov, G. Smith, D. Neumann, N. Wirström, E. Rosenberg, and M. Villa, *Sorma — business cases for an open grid market: Concept and implementation*, in GECON '08: Proceedings of the 5th international workshop on Grid Economics and Business Models, Berlin, Heidelberg, 2008, Springer-Verlag, pp. 173–184.

[16] Optimis, *Optmised infrastructure services*. World Wide Web electronic publication. $http : //www.optimis - project.eu/$ accessed 04-Jan-2013.

[17] S. Sehgal, M. Erdélyi, A. Merzky, and S. Jha, *Understanding application-level interoperability: Scaling-out mapreduce over high-performance grids and clouds*, Future Generation Comp. Syst., 27 (2011), pp. 590–599.

[18] A. W. Services, *Amazon elastic compute cloud (amazon ec2)*. World Wide Web electronic publication. $http : //aws.amazon.com/ec2$ accessed 07-Oct-2010.

[19] K. Stanoevska-Slabeva, D. M. Parrilli, and G. Thanos, *Beingrid: Development of business models for the grid industry*, in GECON '08: Proceedings of the 5th international workshop on Grid Economics and Business Models, Berlin, Heidelberg, 2008, Springer-Verlag, pp. 140–151.

[20] Sun, *Sun grid engine*. World Wide Web electronic publication. $http : //wikis.sun.com/display/GridEngine/Home$ accessed 07-Oct-2010.

[21] M. Surridge, S. Taylor, and D. D. Roure, *Experiences with gria - industrial applications on a web services grid*, in In E-SCIENCE 05: Proceedings of the First International Conference on e-Science and Grid Computing, IEEE Computer Society, 2005, pp. 98–105.

[22] K. Wasielewska, M. Ganzha, M. Paprzycki, M. Drozdowicz, D. Petcu, C. Badica, N. Attaoui, I. Lirkov, and R. Olejnik, *Negotiations in an Agent-based Grid Resource Brokering System*, Saxe - Coburg Publications, 2012, ch. 16, pp. 355 – 374.

[23] XtreemOS Consortium, *Methodology and design alternatives for federation and interoperability*, in XtreemOS public deliverables - D3.5.15, A. Arenas, ed., Work Package 3.5, March 2010.

[24] J. Zhang and J. Luo, *Agent based automated negotiation for grid*, in Computer Supported Cooperative Work in Design, 2008. CSCWD 2008. 12th International Conference on, april 2008, pp. 330 –336.

Edited by: Enn Õunapuu and Vlado Stankovski
Received: Dec 27, 2012
Accepted: Jan. 04, 2013