Efficient Resource Virtualization and Sharing Strategies for Heterogeneous Grid Environments

Paweł Garbacki
Delft University of Technology
Delft, The Netherlands
Email: p.j.garbacki@tudelft.nl

Vijay K. Naik
IBM T. J. Watson Research Center
Yorktown Heights, NY, USA
Email: vkn@us.ibm.com

***Abstract*— Resource virtualization has emerged as a powerful technique for customized resource provisioning in grid and data center environments. In this paper, we describe efficient strategies for policy-based controlling of virtualization of the physical resources. With these strategies, virtualization is controlled taking into account workload requirements, available capacities of physical resources, and the governing policies. Realizing this control requires simultaneous handling of three problems: (i) determining the virtual resource configurations, (ii) the mapping of resulting virtual resources to physical resources, and (iii) the mapping of workloads to the virtual resources. We pose this as an optimization problem and solve this problem using a linear programming (LP) based approach. We evaluate this approach by implementing it in the Harmony grid environment consisting of heterogeneous resources and heterogeneous workload. Experimental results indicate that our approach is efficient and effective. We extend this approach further by using a two-phase heuristic that allows the decision making component to scale up to handle large scale grid systems.**

I. Introduction

Grid systems, in particular those of large scale, are composed of resources belonging to multiple administrative domains. Resources from different administrative domains tend to be heterogeneous, at least in terms of their capacities and performance, and often in terms of OS platforms. Unless this heterogeneity can be masked from the grid workload[1], grid resources remain fragmented and cannot be used as one large cohesive set of resources, thus defeating the purpose of forming the grid in the first place.

Aside from heterogeneity, multiple administrative domains result in resource specific governing policies. As a result, the availability of grid resources cannot always be guaranteed nor is their availability always predictable. However, for grid systems to become viable in enterprize environments, resource availability issues must be addressed transparent to the grid users. Grid users are willing to trade some performance degradations if the resource management issues are handled transparently by the grid management middleware. This means, when a physical resource becomes unavailable, the grid resource management system is expected to provision for alternate resources or migrate the workload handled by the affected resource to another available grid resource.

Recently, resource virtualization [1], [2] is being increasingly considered to address some of the above mentioned grid specific requirements [3], [4]. The key advantages of virtualization technology, in the context of grid systems, are (i) the ability to share physical resources in a controlled and precise manner, (ii) the ability to migrate, (iii) the ability to customize the resource environment to meet workload requirements, and (iv) the ability to control the virtual resource execution and thus, allowing better policy enforcement.

Resource virtualization facilitates controlled and precise sharing of underlying physical resources such as processor cycles, memory units, and disk space. With virtualized resources, service instances are not directly deployed on top of specific physical resource instances. But instead are deployed on a collection of virtualized resources including virtualized processor, memory, filesystem, network, and so on. These virtualized resources are mapped to physical resources transparent to the services deployed in the virtual machine (VM). In case of a policy change or unexpected capacity degradation, e.g., due to a hardware failure, an entire virtual resource can be moved from one physical resource instance to the other. In most VM technology implementations, the VM migration across resources essentially boils down to copying of a VM image file from one physical resource to another. Similarly, virtualization technology allows a service to customize its execution environment without affecting the execution environments of other services. Services with incompatible resource configurations or conflicting co-allocation requirements can still use the same physical resource instance as long as they use different virtual resources. Finally, resource specific policies can be more easily enforced at the virtual resource level. For example, by controlling the physical capacities assigned to virtual resources, it is possible to control the resource capacities used by services running inside a VM even if such a facility is not provided by the OS platform.

The virtualization technology, however, cannot be used "out of the box". The performance degradation caused by running workloads on virtualized instead of bare grid resources, the VM instantiation cost, and additional storage capacity needed to host VM images requires judicious use of virtualization.

Focus of this paper is to describe the decision making com-

[1]In this paper, by workload we mean a set of service requests, transaction requests, jobs, etc. Services or jobs within a workload may be run simultaneously or individually or in some other combination. For naming consistency, in the rest of the paper we shall refer to a workload as a composition of service requests that are executed by service instances

1-4244-0799-0/07/$25.00 ©2007 IEEE

ponent associated with the management layer used to manage and control the virtualized resources. The decision making component takes into account: (i) the costs associated with resource virtualization, (ii) available physical resources, their capacities and associated policies, (iii) the current demand of the grid workload, and (iv) existing VM configurations and instantiated services. Based on these parameters, we determine the virtualization strategy that is optimal according to the current system objectives. Depending on the tradeoffs, the virtualization strategy may result in using existing VMs to run new workload (by increasing the level of sharing), configuring and deploying of new VMs, or migrating of existing VMs. We also describe the implementation of our approach in the context of Harmony grid environment — a VM-based grid system developed at IBM T. J. Watson Research Center [5], [6]. Our implementation is evaluated using real-world data to model the grid environment.

The rest of the paper is organized as follows. In Section II we describe the virtual resource allocation problem considered in this paper. In Section III we present an algorithm that finds the optimal solution of the virtual resource allocation problem. Section IV extends the optimal algorithm with an execution-time-efficient heuristic. In Section V we present integration of our algorithms with Harmony. Section VI describes the results of the performance evaluation of our algorithms. Section VII gives an overview of the related work. Finally, Section VIII concludes the paper.

II. Problem Setting

We consider a grid environment that provides a platform for execution of customized services. A task executed in batch mode or invocation of a transaction in response to a request are examples of services provided by the platform. Multiple instances of the same service may manifest at the same time. Services are not instantiated directly on physical resources, but they are embedded inside VMs.

Although several projects [3], [4], [7] advocate resource virtualization as the right approach to service workload execution in grid environments, they all assume that services are executed in private VMs. However, deploying a separate VM for each service is an expensive and inefficient proposition because of the VM memory overheads and VM instantiation costs. Similarly, because of the VM image size and possibly large number of different VM configurations required to satisfy various requests, it is not practical to replicate VM images on all physical machines. Instead they are stored in a central repository. Transferring a virtual machine image from the image repository to the hosting machine consumes significant network bandwidth, which can lead to bottlenecks at the image repository. In the extreme case of services with low resource requirements and short execution time, the VM instantiation itself can consume more resources than the deployed service. Hosting of a VM also has overheads. In addition to the disk space consumed by the VM image, the OS processes and demons in the VM consume memory and CPU cycles of the host.

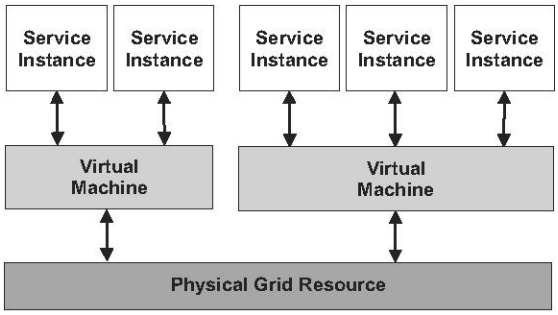

Fig. 1. Sharing of virtualized grid resources.

The costs associated with resource virtualization can be decreased by enabling sharing of VMs among the services (see Figure 1). A virtual resource is shared, if two or more services use the same VM. Obviously, VM sharing also has its limitations. The isolation benefits disappear when services are placed inside the same VM. Furthermore, service specific policies may prohibit co-deployment of some services in the same VM. Such policies are common in environments where services may process sensitive data. In such cases, resource virtualization provides the necessary isolation guarantees while sharing the physical resource. Finally, the VM configuration required by one service may not be compatible with the requirements of another service (e.g., services may require VMs to be configured with different operating systems). In short, to benefit the most from resource virtualization, sharing of both the physical and virtual resources should be analyzed simultaneously. Efficient mechanisms are required to determine the allocation and sharing of physical and virtual resources such that global system objectives are achieved while respecting service-specific and resource-specific policies. We refer to this as the *resource virtualization problem* and, in this paper, we propose a systematic approach for solving the problem. Our approach guarantees optimality of the solution according to a set of global system objectives. Specifically, the problems addressed in this paper are:

- determining the optimal number of VMs to instantiate at any given time,
- the configuration of each VM including the amounts of physical capacities assigned to the VM and the set of software components to be installed inside the VM to satisfy the requirements of the embedded services,
- the mapping of services to VMs such that service specific policies and requirements are satisfied,
- the mapping of VMs to physical resources respecting physical capacity constraints, resource specific policies, and global system objectives.

We note that our approach is also applicable to resource matching in a grid environment where the virtual resource layer is absent or consists of a single VM for each service, although in those cases more straightforward approaches exist [8], [9].

III. Modeling the Virtualization Problem

In the following, we first describe a model of the grid environment. Using this model, the problem described in Section II is then modeled as an optimization problem in terms of linear programming formulation. The solution of the linear program provides an optimal strategy for the controlling resource virtualization.

A. Model of a Grid Environment

The grid environment consists of *resources* that can be either hardware configurations or software packages. Resources are assigned *types* such as server machine, database, network subsystem, file server, etc. Each type is attached a set of *attributes* with values specific to the resource instances of that type. Attribute values can be either immutable, as in case of *static attributes*, or they can be affected by the set of VMs assigned to the resource instance, as in case of *dynamic attributes*. Some examples of static attributes for resource type server are: IP address, CPU architecture, number of CPUs. The set of server's dynamic attributes contains: amount of available memory, free disk space, CPU utilization.

For deployment, each service requires a set of VMs deployed on resources of certain types with specific values of the static attributes and sufficiently large values of the dynamic attributes. A service can be deployed only if all of its requirements are satisfied by the set of assigned (virtualized) resources. By assigning a service to a resource we assume that a certain amount of resource capacities, specified individually for each service, is consumed.

Since large-scale grid environments usually span multiple administrative domains governed according to local rules, we allow each resource instance to define specific *usage policies*. A simple resource policy may, e.g., allow the resource to share its capacity only during certain hours. More complex policies can condition the access to the resources based on the service properties, e.g., by giving priority to VMs running local services deployed by users that belong to the same administrative domain as the resource. Analogous to resource usage policies, we allow services to specify *isolation requirements*, prohibiting isolated services to share the same VM. Service isolation provides protective measures against malicious services that try to affect correct execution of other services.

In addition to resource usage policies and service preferences, we define global objectives that allow us to compare the quality of allocations in terms of fulfilling system-wide goals. Some examples of global objectives include maximizing the throughput, balancing of load across resources, minimizing the number of resources used.

B. Notation

We model the virtual resource allocation problem as a linear program (LP) [10]. Linear programming has been extensively investigated in the literature and a variety of solvers for these optimization problems has been proposed [11]. Before presenting the set of linear expressions that describe the virtual resource allocation problem, we introduce some notation. Our linear program takes the following input parameters describing services and grid resources:

- S is the set of services (service instances),
- R is the set of physical resources,
- T is the set of physical resource types,
- A is the set of dynamic resource attributes. Attributes are unique across resource types — resources of different types are assigned distinct attributes,
- $N_{(r,a)}$ is the capacity of attribute a, $a \in A$ of resource r, $r \in R$,
- $U_{(s,a)}$ is the capacity of attribute a, $a \in A$ consumed by service s, $s \in S$,
- $E_{(s,t)}$ is the set of resources of type t, $t \in T$, with static attribute values satisfying requirements of service s, $s \in S$.

The above set of parameters provides a formal description of the considered grid environment. In this environment the properties of grid resources are defined by specifying capacities of their attributes ($N_{(\cdot,\cdot)}$). E.g., a resource representing a server may define capacities for the attributes describing available memory and CPU cycles. Services describe dependencies on resources by specifying the consumptions of the attribute capacities ($U_{(\cdot,\cdot)}$). As described in Section III-A, in addition to resource capacity consumptions, a service specifies the required values of the static resource attributes. E.g., some services can run only on a server located in a certain network domain. Based on the observation that resources with required values of the static attributes can be identified in a preprocessing step, we do not include the static-attributes-related requirements in the set of input parameters. Instead, we define for each service and dependent resource type a set of resource instances of that type with required values of the static attributes ($E_{(\cdot,\cdot)}$).

The following parameters relate to the virtual resource layer:

- V is the set of virtual machines,
- Q_a is the capacity of attribute a, $a \in A$, consumed by a VM in addition to the consumption of the services running inside that VM. We assume that the attribute capacity consumption is the same for all VMs,
- $I_{(s_1,s_2)}$ equals 1 if isolation requirements allow services s_1 and s_2, $s_1, s_2 \in S$, to run inside the same VM,
- $Y'_{(v,r)}$ equals 1 if virtual machine v, $v \in V$ is currently deployed on physical resource r, $r \in R$; equals 0 otherwise,
- $M'_{(s,v,r)}$ equals 1 if service s, $s \in S$, is currently instantiated inside virtual machine v, $v \in V$, which is, in turn, deployed on physical resource r, $r \in R$; equals 0 otherwise. Note that $M'(s,v,r)$ equal to 1 implies that also $Y'_{(v,r)}$ equals 1,
- C_{srv} is the cost of creating, removing or modifying of a configuration of a service inside a VM. We assume that this cost is constant for all the services and VMs,
- C_{vm} is the cost of creating, modifying or changing of a VM configuration on a physical resource. We assume

that this cost is constant for all the VMs and physical resources.

The set of virtual machines (V) represents VMs that already exist in the system as well as new VMs that will be assigned configurations and physical resources during the matching process. As explained in Section II, hosting of a VM itself incurs a non-zero cost. We express this cost as resource capacity consumption in addition to the resource consumptions by the services configured inside the VM (Q). Service isolation requirements ($I_{(\cdot,\cdot)}$) restrict service co-allocations in the same VMs. Existing mappings of VMs to physical resources ($Y'_{(\cdot,\cdot)}$) and services to VMs ($M'_{(\cdot,\cdot,\cdot)}$) are also included in the set of input parameters. As motivated in Section II, there are non-negligible costs involved in configuring a service (C_{srv}) and a VM (C_{vm}).

In addition to the input parameters, we define a set of output variables that store the solution of the matching problem:

- $Y_{(v,r)}$ is a 0/1 variable equal to 1 if virtual machine v, $v \in V$ has been mapped to resource r, $r \in R$,
- Z_s is a 0/1 variable set to 1 if service s, $s \in S$, has been assigned all the required resources,
- $M_{(s,v,r)}$ is a 0/1 variable set to 1 if service s, $s \in S$, has been assigned virtual machine v, $v \in V$, that has been mapped to resource r, $r \in R$. Note that $M_{(s,v,r)}$ equal to 1 implies that also $Y_{(v,r)}$ is equal to 1,

Note that the mappings of services to VMs and VMs to physical resources can be deduced from the values assigned to $M_{(\cdot,\cdot,\cdot)}$. Thus, the values of $M_{(\cdot,\cdot,\cdot)}$ are sufficient to represent a solution to the resource virtualization problem. The remaining (auxiliary) variables are introduced only for simplifying the problem formulation as a linear program.

C. Constraints

The feasibility of a particular assignment of values to the output variables in the context of the virtual resource allocation problem is determined by a set of constraints. All constraints in the linear program have to be linear expressions.

1) Gang Matching Constraints: Gang matching constraints ensure that we allocate to a service either all resources requested by that service or none of the requested resources. Formally, we can write this requirement as:

$$\sum_{v \in V, r \in E_{(s,t)}} M_{(s,v,r)} = Z_s, \tag{1}$$

for all services $s \in S$ and dependent resource types $t \in T$.

Since $M_{(\cdot,\cdot,\cdot)}$ and Z are 0/1 variables, gang matching constraints guarantees that Z_s equals 1 only if, for a fixed service and a fixed resource type, exactly one of the variables $M_{(\cdot,\cdot,\cdot)}$ is equal to 1. Z_s is, thus, set to 1 only if service s is assigned to exactly one VM deployed on a resource of each of the dependent types.

2) Resource Capacity Constraints: The resource capacity constraints guarantee that the consumption of the resource capacities does not exceed the total available capacity declared by the resources. Two factors determine the consumption of the capacities of a physical resource: the VMs deployed on the resource, and the service instances configured in these VMs. Hence, the resource capacity constraints translate to the following set of linear expressions defined for each resource $r \in R$ and its dynamic attribute $a \in A$:

$$\sum_{s \in S, v \in V} U_{(s,a)} * M_{(s,v,r)} + \sum_{v \in V} Q_a * Y_{(v,r)} \leq N_{(r,a)}. \tag{2}$$

3) Service Isolation Constraints: Two services can be placed inside the same VM only if it is not against their isolation requirements. For all services $s_1, s_2 \in S$, virtual machines $v \in V$, and resources $r \in R$:

$$M_{(s_1,v,r)} + M_{(s_2,v,r)} \leq 1 + I_{(s_1,s_2)}. \tag{3}$$

$M_{(s_1,v,r)}$ and $M_{(s_2,v,r)}$ are both equal to 1 if services s_1 and s_2 run inside the same VM. This is possible only if the value of $I_{(s_1,s_2)}$ is also 1.

4) VM Deployment Constraints: All VMs that are assigned services have to be deployed on physical resources:

$$M_{(s,v,r)} \leq Y_{(v,r)}, \tag{4}$$

for each virtual machine v, $v \in V$, service $s \in S$, and resource $r \in R$.

5) Resource Allocation Uniqueness Constraints: Every virtual machine $v \in V$ is deployed on at most one physical resource:

$$\sum_{r \in R} Y_{(v,r)} \leq 1. \tag{5}$$

Not every VM has to be instantiated. In particular, to minimize physical resource consumption, a VM that does not embed any service need not be assigned a physical resource. We consider this requirement to be part of the objectives of the optimization problem.

D. Optimization Objectives

The objective function defines the quality of a virtual resource allocation when multiple feasible solutions exist. Virtual resource allocation algorithm uses objective function to select the best among the feasible solutions.

The virtual resources layer increases the number of service placement possibilities, consequently allowing more expressive objective functions to be defined. The common denominator for these objective functions is that they all have to take into account the cost of configuring services and VMs as well as the cost of migrating existing service and VM configurations.

The cost C_1 of (re)configuring services in VMs equals:

$$C_1 = \sum_{s \in S, v \in V, r \in R} C_{srv} * |M_{(s,v,r)} - M'_{(s,v,r)}|. \tag{6}$$

Note that services assigned to the same VMs deployed on the same physical resources as during the previous virtual resource allocation do not incur any additional cost. Services that have just been added to the system affect C_1 with the configuration cost C_{srv}. For services that have to be reconfigured inside a VM (VM_{new}) other than the one where they are currently deployed ($VM_{current}$), C_1 is increased by $2 * C_{srv}$, which is

the cost of migrating the service from $VM_{current}$ to VM_{new}. Cost C_1 also captures the case where the service is deployed in the same VM as previously, but the physical resource of that VM has changed. VM migration impacts the service specific cost since during VM migration embedded services have to be suspended.

The cost C_2 of (re)deploying of VMs on resources can be expressed as:

$$C_2 = \sum_{v \in V, r \in R} C_{vm} * |Y_{(v,r)} - Y'_{(v,r)}|. \quad (7)$$

Similarly, as in the case of service reconfigurations, the cost associated with restructuring of the virtual layer depends on the modifications required to the current structure of this layer. Cost C_2 is independent of the services configured inside VMs. All service specific costs are included in the value of C_1.

The global optimization objective is composed of costs C_1, C_2, and a *customized objective function* O defining the current matching objectives that do not relate to the virtualization costs. As an example, consider the objective function of maximizing the throughput, i.e., the number of service requests processed per unit time:

$$O = \sum_{s \in S} Z_s. \quad (8)$$

In [8], we define linear constraints for customized objective functions optimizing the prioritized throughput, the number of resources used, the load balance, and service preferences on resources. Any of those objective functions can be easily combined with the cost functions as:

$$maximize \quad (O + k_1 * C_1 + k_2 * C_2). \quad (9)$$

The constants k_1 and k_2 control the impact of the cost functions on the value of the global optimization objective.

IV. On-line Resource Matching

In this section we extend the requirements of the virtual resource allocation (matching) problem by introducing constraints on the amount of time required to decide on the service placements. We address those additional requirements by combining the LP-based virtual resource allocation algorithm presented in Section III with a time-efficient heuristic method of finding suboptimal allocations. The matching heuristic decomposes the virtual allocation problem into two subproblems which are solved independently by an algorithm that provides (suboptimal) solutions in a designated amount of time.

A. On-line Matcher

The linear program formulated in Section III can be classified as a 0-1 integer program (IP) since the optimization variables are all required to be 0 or 1. Although the sophistication of the existing linear solvers allows finding the solution of a 0-1 integer program very efficiently for most of the problem instances, solving of a 0-1 IP is in general NP-hard.

The negative impact of the possibly long virtual resource allocation time escalates if some of the services are attached hard deadlines. To address this problem, we propose a resource matching heuristic that makes instant decisions regarding virtual machine allocation. The heuristic approximates the optimal virtual resource allocation by considering only some of the matching objectives. Since no guarantees about the optimality of the matchings found by the heuristic are provided, we periodically run the LP-based resource allocation algorithm presented in Section III to reduce the possible drift from the optimum. Thus, the on-line matcher combining the heuristic with the LP-based resource allocation algorithm can keep up with a high arrival rate of the matching requests while maintaining high quality of the virtual resource allocations.

B. Heuristic Approach to Resource Matching

The heuristic resource matching is performed in two steps (phases). During the first phase we select for each service a VM(s) where it will be deployed. The objective of the second phase is to map those VMs to physical resources.

1) First Phase: In the first matching phase we take into account only the services that are not assigned to any VM and VMs that are deployed on physical resources. For each VM and each dynamic attribute of the resource where the VM is deployed, we define a *virtual attribute* with capacity equal to the available capacity of the physical attribute. Note that since more than one VM can run on a single physical resource, the aggregate capacity of the virtual attributes can exceed the total available capacity of the corresponding physical attribute. Once the values of the virtual attributes are defined, the matching of services with the virtual resources is performed. During the matching, service requirements are compared against the virtual attribute values.

The way the virtual attribute values are defined does not guarantee that, after the matching, the physical capacities will be preserved. To bring the system back to a consistent state, we iterate over the mappings found by the matching algorithm and invalidate those that violate the physical resource capacity constraints.

If no existing virtual resources have been allocated to a particular service, we logically (without assigning it to a physical resource) create a new VM with a configuration conforming to the embedded service requirements. The output of the first phase of the resource matching heuristic algorithm is, thus, a set of mappings of services to the already instantiated as well as logical VMs.

2) Second Phase: In the second phase of our resource matching algorithm, we allocate physical resources to the new VMs that have been logically created during the first phase. The physical resources are selected based only on the VM specifications defined in the first phase. Further no knowledge of the requirements of individual services is needed in the second phase.

3) Time-Constrained Matching: The decomposition of the resource matching into two phases executed separately decreases the complexity of the virtual resource allocation problem. On the baseline, the matching problems considered in these two phases are similar. While in the first phase, matching

takes into account service-specific requirements and capacities of the VMs, in the second phase VM-specific requirements and physical resource capacities are considered. The only difference in the matching problems of the two phases lies in the policy interpretation aspect. The objective of the first phase is to produce matchings that conform to service policies. In the second phase, service policies are replaced by the resource-specific policies. Any of the standard grid resource matching algorithms that accepts the policy descriptions passed as the parameters of the resource matching model and additionally provides execution time guarantees, can be used to solve the matching problems of the two phases. Some examples of grid resource matching algorithms that satisfy these criteria have been described in [8], [9].

V. Integration with Harmony

In our previous work we have developed Harmony, a platform for delivery of customized services configured on virtualized grid resources [5], [6]. In this section we describe the integration with Harmony of the virtual resource allocation mechanisms introduced in Sections III and IV. We first present the highlights of the Harmony architecture. Then, we describe the method of extracting resource requirements from service workloads. Finally, we describe the implementation of the virtual resource allocation mechanisms in Harmony.

A. Overview of the Harmony Architecture

The architecture of Harmony is defined using a layered approach presented in Figure 2. The components of our architecture can be divided into two functional blocks, namely Service Provisioning Infrastructure and Service and Resource Management Infrastructure.

1) Service Provisioning Infrastructure: The Service Provisioning Infrastructure consists of four layers that represent service access point, service instances, VMs embedding the service instances, and the physical grid resources hosting VMs.

System users and applications that invoke the services are collectively called Service Clients. Access Layer is represented by Gateway which is a well known access point where service clients direct their requests. Gateway reroutes the client requests to service instances where the requests are processed. Request routing is fully transparent to the service clients — clients do not have any influence on, or knowledge of which service instance handles their requests.

Service instances, collectively forming the Service Layer, are not running directly on the physical resources, but are rather embedded inside VMs. Multiple service instances may be placed inside different VMs and a single service instance may require multiple VMs. Depending on the policies defined by the services and compatibility issues between service configurations, multiple service instances may reside inside a single VM.

The virtualized resources and the associated control infrastructure form the Virtual Resources Layer. Every VM is controlled by the Virtual Machine Manager which runs as a privileged process inside the VM. Virtual Machine Manager coordinates service instantiation and monitors the CPU, memory, disk, and bandwidth usage of the configured service instances.

Physical Resources Layer represents the grid resources. Physical Machines may join and leave this layer dynamically. Typically, the resource availability schedule is governed by a set of policies defined by the resource owner. The counterpart of the Virtual Machine Manager for physical resources is the Host Agent. The Host Agent runs in a demon mode on each physical resource, monitoring the CPU, memory, disk space, and network bandwidth usage of the VMs, ensuring that none of the local policies is being violated. The task of instantiating VMs on physical resources is assigned to the Virtual Machine Instantiator.

2) Service and Resource Management Infrastructure: Fulfilling certain QoS requirements, while respecting resource usage policies, requires coordination of the management decisions at different layers of the Resource Provisioning Infrastructure. For example, configuring more service instances improves the client request throughput but also increases the resource capacity consumptions which can exceed the policy-allowed limits. The integration of the management across the layers of the Resource Provisioning Infrastructure is realized by the components of the Service Management Infrastructure.

The Active State Repository gathers the virtual and physical resource usages measured by the monitoring components (Virtual Machine Manager and Host Agent). The individual measurements are correlated with each other to produce higher level statistics, e.g., describing for each service the aggregate resource usage of all existing instances of this service.

Predictor generates forecasts of the future service workload and resource availability based on the current system state as well as historical data. On-line Resource Matcher decides on the structure of the bindings between service instances, VMs and physical resources. Predictor and On-line Resource Matcher are extensively described further in the paper.

To provide service agility, Harmony system has an automated service instance configuration feature that allows dynamic migration of service instances to the point of resource availability. In Harmony, Configuration and Deployment Engine customizes the process of instantiating new VMs. It also installs and configures services and dependent software components inside those VMs.

The Grid Resource Manager (GRM) deals with the high level QoS guarantees. The objective of GRM is to guarantee that there are enough resources allocated to services to meet certain QoS requirements, while ensuring that the service workload does not violate the resource usage policies.

B. Identifying Service Requirements

The capacity requirements of service instances depend on the client demands. Demands are constantly changing over time [12]. Service resource requirements cannot be, thus, predefined, but they have to be extracted dynamically during the system operation. We present here a method of determining

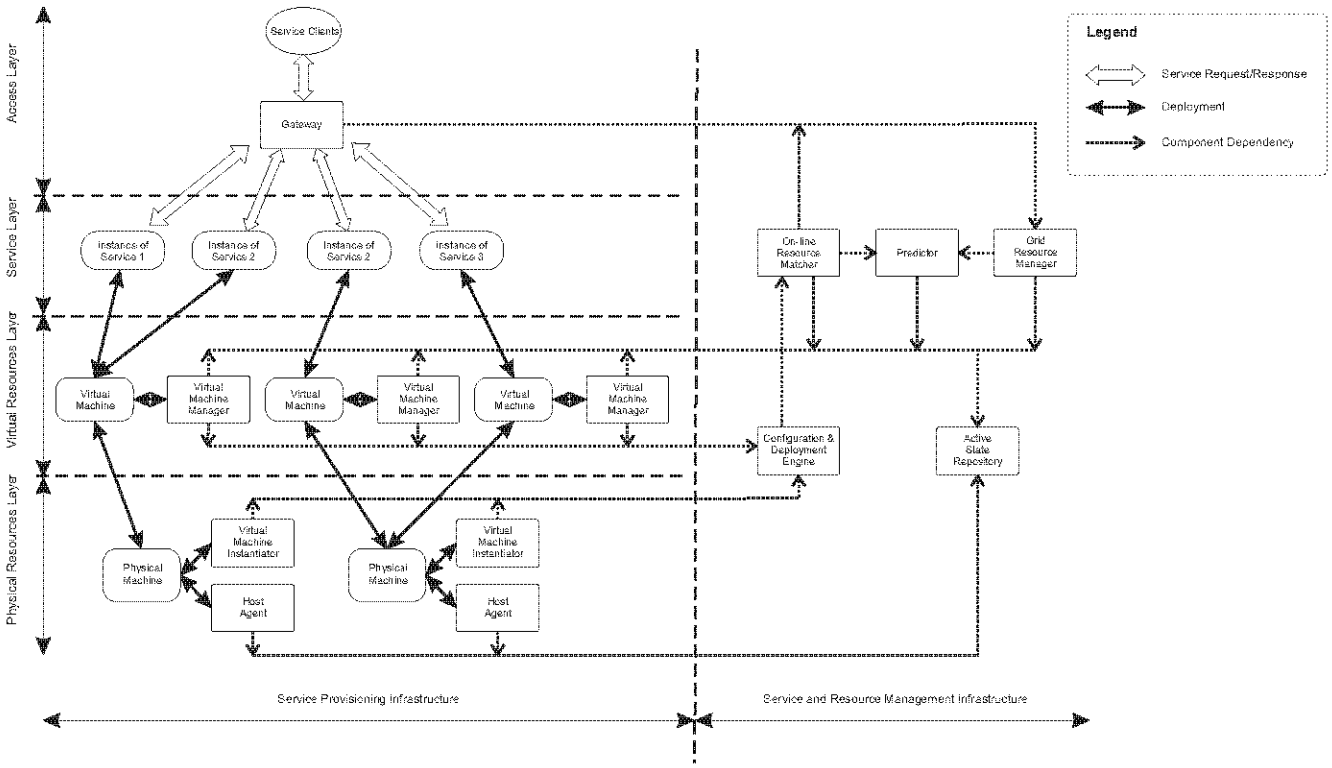

Fig. 2. Harmony architecture.

the resource capacities required by individual services based on resource usage predictions. To decrease the amount of processed data, we aggregate service requests over fixed-length time intervals before applying the prediction algorithm.

Instead of modeling workload demands at a single request resolution, we use prediction methods to identify longer term trends in the service invocation patterns. These trends are computed for each service separately. Trend provides a basis for the estimation of the client demand for a particular service. The demand, in turn, directly translates to the resource capacities required to satisfy this demand. The required resource capacities are considered while allocating virtual and physical resources to service instances.

Predicting of grid resource usages has been recognized as a difficult problem [13]. In particular, there is no single prediction algorithm that fits all workloads. Having the generality of our design in mind, instead of supporting a single prediction algorithm, we use a wide range of forecasting algorithms, starting with simple methods such as running mean or exponential smoothing, to end up with current state-of-the-art approaches such as ARIMA, Holt Winters, FFT, Wavelet, or Kalman Filter [14]. For each prediction method, we measure its accuracy in a certain context, e.g., we estimate the load exercised by clients of a particular service, and select the most reliable method for this context. In this respect, our prediction approach is similar to the one adopted in the Network Weather Service [15]. For a detailed description of workload characteristics prediction in Harmony we refer to [5].

The possibly short execution time of a service request, resulting in a high number of requests per time interval, makes service requests expensive to analyze individually. Furthermore, the execution time of a single request does not provide a reliable estimate of the long-term behavior of a typical service that usually exhibits a high level of burstiness on short-term time scales [12]. When it comes to selecting the resources for service instance deployments, the longer-time estimates are more relevant. This is due to the fact that deploying services is an expensive operation [5]. The lifetime of the instance should be, thus, long enough to amortize the instantiation cost.

To support services with many short requests efficiently, we aggregate multiple requests over predefined time intervals. The length of the *aggregation interval* controls the granularity of the predictions. The longer the aggregation interval, the higher is the discrepancy between the predicted and the actual load in this interval. On the other hand, longer aggregation intervals result in more stable deployments, decreasing the overhead incurred by restructuring on the Service and Virtual Resources layers (see Figure 2).

Finding the optimal length of the aggregation interval requires considering several properties of the workload execution environment. First, the characteristics of the workload itself influence the aggregation method. High fluctuations of client demands provide rationale for shorter aggregation intervals that will better cope with the frequent changes in the demand. Second, the set of policies specifying the resource usage rules

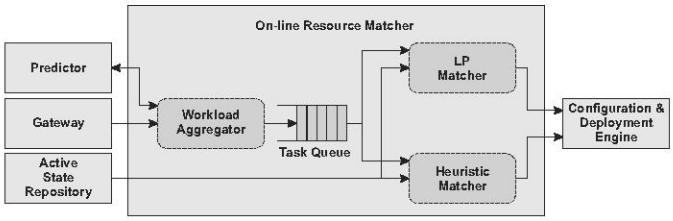

Fig. 3. Dataflow between the On-line Resource Matcher and Harmony components.

and workload orchestration guidelines can impose implicit bounds on the length of the aggregation interval. E.g., resource usage policies allowing the resource instance to be used by a particular service only for a certain amount of time disposes aggregation intervals longer than that amount of time. Finally, the cost of the deployment and configuration of a new service instance should be taken into account while performing the workload aggregation. Shorter aggregation intervals motivated by high fluctuations in client request patterns lead to more redeployments and reconfigurations of the service instances.

C. Implementation of the On-line Resource Matcher

The method of identifying service requirements presented in Section V-B provides a prerequisite for the virtual resource matching. We have implemented the on-line resource matching approach described in Section IV and integrated it with the Harmony infrastructure. The functionality of allocating the virtual resources is provided in Harmony by the On-line Resource Matcher. The On-line Resource Matcher is logically divided into several components that interact with each other and external Harmony components as presented in Figure 3. An arrow in Figure 3 indicates a dataflow direction.

Service client requests are reported by the Gateway at *Workload Aggregator*. Workload Aggregator analyzes the requests at a granularity determined by the aggregation interval. Predictor is involved in workload analysis, helping to identify patterns in service invocation schemes. Workload Aggregator is implemented as a web service which makes its functionality easily accessible for the Gateway.

Service requirements arising from the workload aggregation are appended to the *Task Queue*. Depending on the current system load, the matching is performed by either the *LP Matcher* or the *Heuristic Matcher*. The extent of the system load is determined by the length of the Task Queue. If the current length of the Task Queue is lower than a predefined threshold indicating that the system is lightly loaded, then the virtual resource allocation is performed by the LP Matcher. If, however, the size of the Task Queue increases over the threshold, the Heuristic Matcher is activated. Heuristic approach aids the LP algorithm in processing of the matching tasks until the size of the queue drops below the threshold. Note that during the activity period of the heuristic algorithm, the LP matching is also performed and the possible divergence from the optimal resource allocation strategy caused by the inaccuracies of the heuristic is corrected by the LP Matcher execution.

Resource Type	Resource Static Attributes	Resource Dynamic Attributes	Number of Instances
server	CPU architecture, # CPUs, domain	utilization, memory	50
database	vendor	connections	50
network	IP, protocol	bandwidth	50
file storage	filesystem	size	50

TABLE I
RESOURCE MODEL.

The Task Queue contains only the description of the workload characteristics. The specification of the system resources, VMs and configured service instances is provided by the Active State Repository. Service capacity requirements extracted from the workload in combination with the current system configuration provides the complete description of the virtual resource allocation problem.

Our implementation of the LP Matcher models the virtual resource allocation problem in the GNU MathProg language, which is a subset of AMPL [16], a well established standard among LP languages. The LP solving functionality is provided by the open source GNU Linear Programming Kit [17]. In our implementation of the Heuristic Matcher we use a method based on an Evolutionary Algorithm described in detail in [9]. The evolutionary optimization process can be stopped practically at any time, still producing the best suboptimal solution found until that time. Hard execution time guarantees of the Heuristic Matcher can be, thus, provided.

The resource allocation decisions taken by the On-line Resource Matcher are executed by the Configuration and Deployment Engine. Configuration and Deployment Engine performs the necessary restructuring at the Service and Virtual Resource Layers of the Harmony infrastructure.

VI. PERFORMANCE EVALUATION

In this section we describe experimental evaluation of the virtual resource allocation mechanisms described in Sections III, IV, and V.

A. Experimental Setup

The model of the grid resources used in our experiments is based on real-world traces of a deployed service provisioning infrastructure. Namely, we have obtained the detailed information on the resources hosting IT services of IBM customers. These statistics are provided by the Server Resource Management (SRM) [18] system that reports historical and near real time trends of resources serviced by IBM. Some illustrative examples of such services are described in a series of case studies available for download from IBM e-Business Hosting Services pages [19].

Our model of grid environment consists of 200 resources divided into four types: server, database, network, and file storage. Each resource type is assigned one or two dynamic attributes and one, two or three static attributes. Table I summarizes the resources and their attributes.

In this paper we concentrate on evaluation of the efficiency of the virtual resource allocation mechanisms only. The performance aspects of the workload aggregation mechanisms are

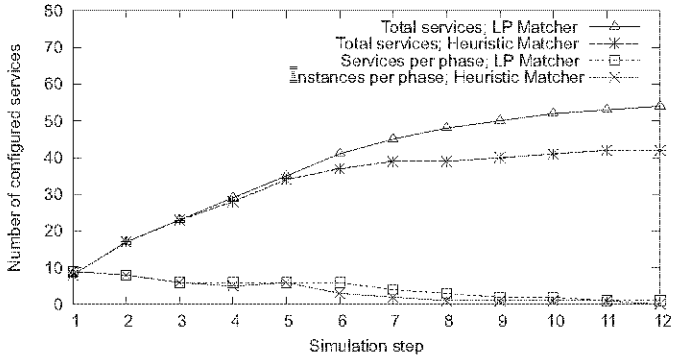

Fig. 4. Comparison of the throughput of the LP and heuristic approaches.

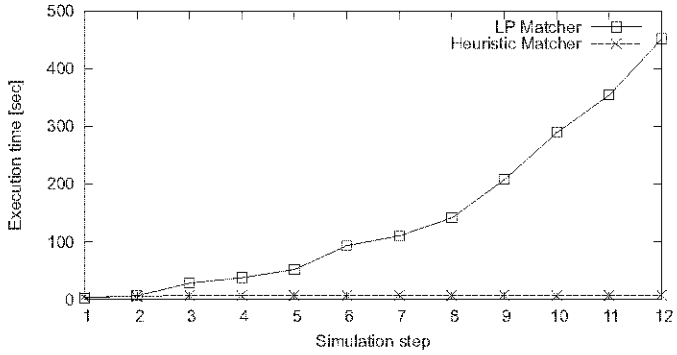

Fig. 5. Execution time of a simulation step for the LP and heuristic approaches.

outside of the scope of this work. On the baseline, the quality of the aggregations depends on the accuracy of the forecasts provided by the Predictor component described in more detail in [5]. We believe that the variety of the forecasting methods implemented in our Predictor component and the presence of mechanisms allowing to dynamically select the best method for a particular workload will cope with the heterogeneity of the workloads. We leave the validation of this claim for the future work.

The workload used in the evaluation is generated synthetically. For each service we select the dependent resource types making sure that each service depends on at least one resource type. The dependency of a service on a resource type is determined by Bernoulli distribution with the probability of success equal to 0.5. After the dependent resource types have been chosen, a set of dependent attributes for each of these types is selected. Each service selects one or more dynamic attributes and zero or more static attributes of the dependent resource type. Also at this stage the selection is performed according to Bernoulli distribution with the probability of success equal to 0.5. The required value of the dependent static attribute is selected randomly and uniformly from the set of available values of this attribute. The minimal required value of the dynamic attribute is selected randomly and uniformly from the interval bounded by 0 and the maximal available value of that attribute among the defined resources.

The objective function that we optimize is the throughput — we maximize the number of configured services while minimizing the cost of modifying the current system configuration, as described in Section III-D. The values of service and VM configuration costs, C_{srv} and C_{vm}, are both set to 0.1 while the parameters k_1 and k_2 are equal to -0.5 giving the cost functions the same weight as the throughput maximization.

B. Experimental Results

Using a series of experiments we compare the quality of the allocations computed by the LP and the heuristic approaches. The experiments are performed in steps. During each step we try to allocate virtual resources for 10 services with requirements generated synthetically according to the method described in Section VI-A. The simulation is repeated 10 times with different random seeds and the average number of services matched in each step is taken.

In our simulation, the resource allocations are preserved between the steps. Consequently, after a number of steps the system becomes congested and no further service instances can be configured unless some other instances are removed. From that point on, only the LP Matcher can lead to any improvement since the heuristic cannot free capacities — it can only add new configurations. Note that in a realistic environment the congestion point is never reached since service instances are removed when the client demand decreases. In our simulation we do not, however, remove the instances, which allows us to investigate how the algorithms perform under different system load conditions.

The congestion point cannot be easily detected as there may always come a service with demand low enough to be satisfied by the available resource capacities. Therefore, we stop our simulation when no improvement (no new matchings found) between two consecutive phases is observed for the Heuristic Matcher.

Figure 4 shows the numbers of service configurations added in a single simulation step and the total number of services assigned resources for each of the matchers. During the first phases both matchers allocate resources for similar number of services. As the system size grows, the LP matcher outperforms the heuristic being able to satisfy the requirements of up to 20% more services.

The better quality of the matching found by the LP approach come at the cost of higher execution time. Figure 5 presents the execution time of each simulation step for both matchers. During the first two steps the execution time of LP and heuristic algorithms is comparable. In the consecutive steps the heuristic algorithm maintains roughly the same execution time while the execution time overhead of the LP Matcher keeps growing.

VII. Related Work

The properties of resource virtualization such as the ease of policy enforcement, ability to provide isolation, facilities for fine-grained resource management, the ability to instantiate independently configured services on a single resource, make

it an attractive enabler of grid computing [3], [4], [7]. The Xenoserver project [20] builds a distributed infrastructure as an extension of the Xen VM [21]. The In-Vigo project [22] proposed a distributed grid infrastructure based on VMs, while the Violin [23] project addresses the virtual networking issues. Although all these projects use VMs to improve the efficiency of resource sharing in grid environments, none of them considers sharing of VMs between multiple workloads or proposes a strategy for determining the optimal allocation of virtual resources.

In [24], authors consider the problem of assigning servers in a data center to application components such that the resulting traffic-weighted average inter-server distance is minimized while satisfying the application resource requirements and network capacity constraints. The resource assignment problem is modeled and solved using mixed integer linear programming formulations. Although this problem is motivated by resource virtualization, their work does not address the two level optimization problem arising in mapping application components to virtual resources and virtual resources to physical resources. The resource matching problem in grid environments has also been studied extensively in [25], [26], [27]. Grid resource matchers satisfying on-line execution time constraints have been described in [8], [9]. All these approaches are, however, limited to traditional grid and data center architectures where services are deployed directly on physical resources. The virtual resource allocation problem presented in this paper is an extension of the traditional resource matching problem in a sense that the traditional problem can be solved using the method introduced in this paper.

VIII. Conclusion

In this paper, we have developed an approach for managing and controlling resource virtualization in the context of grid environments by defining optimal and heuristic strategies for policy-based resource sharing. Virtualization strategies described here take into account the properties of the grid resources, grid workload characteristics, and global system objectives. We have shown that in spite of the complexity and the number of factors that have to be considered while computing the virtualization strategy, it is possible to efficiently find a strategy that is optimal according to some customized objectives. The heuristic algorithm proposed here improves the execution time of the virtualization strategy computation even more, allowing the matching to be performed in an on-line mode. Both the approaches have been implemented and integrated with Harmony — an existing platform for service delivery in grid environments. The experimental evaluation indicates that our approach is able to handle virtualization strategies efficiently. Results presented here help to determine how our solution performs in a realistic environment modeled using real-world grid resource characteristics. Finally, we note that the concepts described in this paper apply to other shared distributed environments such as clusters and data centers, in addition to the shared grid environments.

References

[1] "Virtualization definition from wikipedia." http://en.wikipedia.org/wiki/Virtualization.

[2] IEEE Computer, Special Issue on Virtualization, May 2005.

[3] R. Figueiredo, P. Dinda, and J. Fortes, "A case for grid computing on virtual machines," in *ICDCS'03*, Providence, RI, May 2003.

[4] I. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang, "Virtual clusters for grid communities," in *CCGrid 2006*, Singapore, May 2006.

[5] V. K. Naik, P. Garbacki, and A. Mohindra, "Architecture for service request driven solution delivery using grid systems," in *IEEE International Conference of Services Computing (SCC'06)*, Chicago, IL, September 2006.

[6] V. K. Naik, S. Sivasubramanian, and S. Krishnan, "Adaptive resource sharing in a web services environment," in *Middleware'04*, Toronto, Canada, October 2004.

[7] A. Sundararaj and P. Dinda, "Towards virtual networks for virtual machine grid computing," in *3rd USENIX VM'04*, San Jose, CA, May 2004.

[8] V. K. Naik, C. Liu, L. Yang, and J. Wagner, "Online resource matching for heterogeneous grid environments." in *CCGRID'05*, Cardiff, UK, May 2005.

[9] V. Naik, P. Garbacki, K. Kummamuru, and Y. Zhao, "On-line evolutionary resource matching for job scheduling in heterogeneous grid environments." in *2nd Int'l Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS'06)*, Chicago, IL, July 2006.

[10] A. Schrijver, *Theory of Linear and Integer Programming*. John Wiley & Sons, June 1998.

[11] R. Fourer, "Linear programming software survey," June 2005.

[12] D. A. Menasce, "Workload characterization," *IEEE Internet Computing (Special issue on Grid Computing)*, September 2003.

[13] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley, "Predictive application-performance modeling in a computational grid environment," in *HPDC-8*, Redondo Beach, CA, August 1999.

[14] G. Box, G. M. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting and Control*, 3rd ed. Prentice Hall, February 1994.

[15] R. Wolski, "Experiences with predicting resource performance online in computational grid settings," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 4, March 2003.

[16] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, 2nd ed. Duxbury Press, November 2002.

[17] "GNU linear programming kit page." http://www.gnu.org/software/glpk/.

[18] "SRM page." https://srm.raleigh.ibm.com.

[19] "IBM e-business hosting services." http://ibm.com/e-business/hosting.

[20] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford, "Xenoservers: Accountable execution of untrusted programs," in *HotOS-VII*, Rio Rico, AZ, 1999.

[21] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY: ACM Press, 2003.

[22] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, "From virtualized resources to virtual computing grids: the in-vigo system," *Future Gener. Comput. Syst.*, vol. 21, no. 6, 2005.

[23] P. Ruth, X. Jiang, D. Xu, and S. Goasguen, "Towards virtual distributed environments in a shared infrastructure." *IEEE Computer (Special Issue on Virtualization Technologies)*, 2005.

[24] X. Zhu, C. Santos, J. Ward, D. Beyer, and S. Singhal, "Resource assignment for large-scale computing utilities using mathematical programming," HP Labs, Tech. Rep. HPL-2003-243R1, 2003.

[25] C. Liu, L. Yang, I. Foster, and D. Angulo., "Design and evaluation of a resource selection framework for grid applications," in *HPDC-11*, Edinburgh, Scotland, July 2002.

[26] R. Raman, M. Livny, and M. Solomon., "Policy driven heterogeneous resource co-allocation with gangmatching," in *HPDC-12*, Seattle, WA, June 2003.

[27] X. Bai, H. Yu, Y. Ji, and D. C. Marinescu, "Resource matching and a matchmaking service for an intelligent grid," *Transactions on Engineering, Computing and Technology*, December 2004.