Quartermaster — A Resource Utility System

Sharad Singhal, Sven Graupner, Akhil Sahai, Vijay Machiraju,
Jim Pruyne, Xiaoyun Zhu, Jerry Rolia, Martin Arlitt, Cipriano Santos,
Dirk Beyer, Julie Ward
HP Laboratories Palo Alto
HPL-2004-152
September 9, 2004*

E-mail: {sharad.singhal, sven.graupner, akhil.sahai, vijay.machiraju, jim pruyne, xiaoyun.zhu, jerry.rolia, martin.arlitt, cipriano.santos, dirk.beyer, jward}@hp.com

resource allocation, resource models, policy based composition, capacity planning

Utility computing is envisioned as the future of enterprise IT environments. Achieving utility computing is a daunting task, as the needs of enterprise users are diverse and complex. In this paper we describe Quartermaster, an integrated set of tools and technologies that addresses these needs. Quartermaster supports the entire lifecycle of computing tasks - including the design, deployment, operation, and decommissioning of each task. Although parts of this lifecycle have been explored earlier, Quartermaster integrates all of these parts in a unified framework using model-based automation. All tools within Quartermaster are integrated using models based on CIM, an industry standard meta-model. The paper also discusses our implementation of Quartermaster, and describes several case studies involving HP Customers.

* Internal Accession Date Only
Approved for External Publication

Quartermaster—A Resource Utility System

Sharad Singhal, Sven Graupner, Akhil Sahai, Vijay Machiraju, Jim Pruyne,
Xiaoyun Zhu, Jerry Rolia, Martin Arlitt, Cipriano Santos, Dirk Beyer, Julie Ward

HP Laboratories
1501 Page Mill Road
Palo Alto, CA 94304

{sharad.singhal, sven.graupner, akhil.sahai, vijay.machiraju, jim.pruyne, xiaoyun.zhu, jerry.rolia, martin.arlitt, cipriano.santos, dirk.beyer, jward}@hp.com

Abstract

Utility computing is envisioned as the future of enterprise IT environments. Achieving utility computing is a daunting task, as the needs of enterprise users are diverse and complex. In this paper we describe Quartermaster, an integrated set of tools and technologies that addresses these needs. Quartermaster supports the entire lifecycle of computing tasks - including the design, deployment, operation, and decommissioning of each task. Although parts of this lifecycle have been explored earlier, Quartermaster integrates all of these parts in a unified framework using model-based automation. All tools within Quartermaster are integrated using models based on CIM, an industry-standard meta-model. The paper also discusses our implementation of Quartermaster, and describes several case studies involving HP Customers.

1. Introduction

The increasing complexity of IT environments has made them difficult to manage in a cost effective manner. This has led to a push within enterprises for consolidating IT environments into managed pools of resources. In addition, with the advent of Internet applications with rapidly changing lifecycles, users also require that IT environments provide the ability to rapidly configure and modify applications. As a result, companies such as HP [1], IBM [2], SUN [3], as well as a host of startups [4]-[7] have announced initiatives or products to address these requirements. Many of these initiatives improve IT utilization through various virtualization techniques, while others provide configuration and provisioning capabilities or offer monitoring and management capabilities to manage the IT environment more efficiently.

We define utility computing as the ability to provide complex computing environments on-demand to IT users. The products mentioned above represent the initial efforts in making utility computing a reality. Achieving utility computing is difficult because the needs of enterprise users are complex. Each application running within the enterprise has unique assumptions, each enterprise has different policies that are associated with its applications, and each user brings a different set of requirements to the application. Providing infrastructure that supports these diverse requirements is not a straightforward task. While techniques such as virtualization or load-balancing are necessary for utility computing, they are not sufficient. To be successful, a utility computing system must support design, deployment, and management of arbitrary applications while dealing with their frequently competing requirements for resources; accommodate both user and operator policies on how infrastructure is used; deal with upgrades of both the infrastructure and the applications; and maintain a high level of automation to reduce errors and manage costs.

We address utility computing from the perspective of IT administrators. That is, given user-specified requirements for a complex application, how can the corresponding system specification be automatically created and the system provided to the user on-demand? Our goal is to provide the administrators with tools that support the entire lifecycle of a computing task—including the design, deployment, operation, and decommissioning of that task while maintaining flexibility, agility, and cost efficiency within the utility through automation. Our approach is using model-based automation—creating models of the IT environment and the desired application and creating a set of tools that use those models to automate IT tasks.

In this paper, we describe Quartermaster, an integrated set of tools and technologies targeted at this problem. Quartermaster tools currently provide the following capabilities to IT users and system operators:

- *Policy-based resource composition*: Quartermaster allows operators to capture system composition rules and best practices in models, and provides users the ability to automatically create custom designs that conform to those policies. This reduces the time to design applications and IT environments and reduces the likelihood of error in system design.
- *Capacity management*: Quartermaster includes scheduling and capacity management algorithms that can track complex patterns of time varying resource demands and react accordingly. This enables operators to manage infrastructure use and permits specific qualities-of-service to be achieved.
- *Resource assignment*: Quartermaster uses mathematical programming techniques to ensure that resource-level requirements (e.g., network bandwidth) of the application are met and bottlenecks are not created in the shared infrastructure when resources are assigned to applications. This enables efficient use of the infrastructure resources while ensuring application-level quality of service.

The remainder of the paper is organized as follows: Section 2 describes the functionality provided by Quartermaster, and discusses the various tools that provide those functions. Section 3 presents the Quartermaster software architecture. Section 4 provides two case studies where we are currently using Quartermaster. Section 5 describes related work, and we conclude the paper with a summary of our contributions in Section 6.

2. Quartermaster Tools

Quartermaster integrates a number of tools and technologies that provide system administrators with the ability to create custom designs for applications and to rapidly allocate resources to those applications. Figure 1 shows the overall functional architecture of Quartermaster.

Quartermaster maintains a repository of resource types and instances that it manages. This repository maintains models of the different resources known to Quartermaster, as well as an inventory of resource instances available to Quartermaster. It provides Quartermaster tools with a uniform way of interacting with the resources. IT operators and users can interact with Quartermaster tools using either visual or programmatic interfaces. The tools provide users with the ability to compose IT resources into desired configurations, to manage the capacity of the underlying resource pools, and to schedule and allocate resources to applications. The Quartermaster tools can be integrated with service deployment capabilities offered by other technologies [9] as well as with management tools that provide operations management and control capabilities [10] to provide an end-to-end view of resource management within the data center. In the next few subsections, we describe the capabilities offered by the tools in more detail.

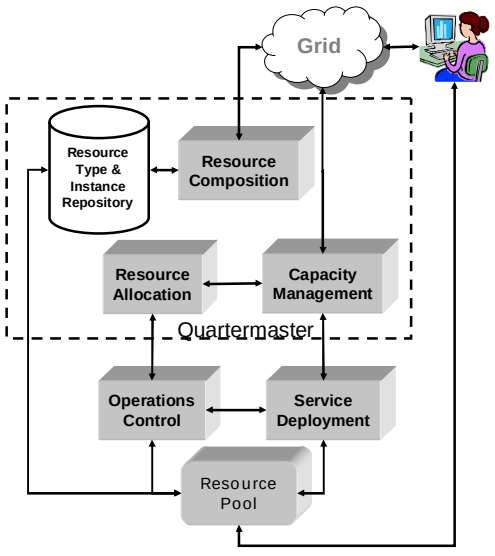


Figure 1: Overall architecture of Quartermaster

2.1. Policy-driven Resource Composition

When resources are combined to form other higher-level resources, a variety of rules need to be followed. For example, when operating systems are loaded on a host, it is necessary to validate that the processor architecture assumed in the operating system is indeed the architecture on the host. Similarly, when an application tier is composed from a group of servers, it may be necessary to ensure that all network interfaces are configured to be on the same subnet or that the same version of the application is loaded on all machines in the tier. To ensure correct behavior of a reasonably complex application, several hundred such rules may be necessary. This is further complicated by the fact that a large fraction of these rules are not inherent to the resources or the application, but depend on preferences (policies) provided by the system operator or indeed, by the user as part of the request itself.

Quartermaster models [8] allow specification of such rules in a distributed manner. By capturing the configuration rules as part of the specification of resource types, and by formalizing how these rules are combined when resources are composed from other resources, Quartermaster provides a very flexible policy-based framework for generating configuration specifications for complex resources. Quartermaster models resources in terms of entities (described later in Section 3). Each entity is characterized by a set of attributes and values taken by those attributes. For resources or applications, the attributes represent configuration or other parameters that are meaningful for operation. For configuration activities, attributes indicate whether a particular activity needs to be triggered by the deployment system, and if so, provide the parameters that are required for that activity. Policies are formulated as logical constraints that restrict values possible for model attributes, and are used during system composition to ensure correctness.

Complex environments requested by users are themselves treated as higher-level resources that are composed from other resources. Policy is embedded in the various resource types, specified by the operators of the resource pool, or by users as part of the requests for resources, and restricts the composition choices used when composing higher-level resources from the component resources. Unlike traditional policy-based systems, the policy model does not couple actions to constraints, and actions (workflows) needed for realizing the specification of the higher level resource can be automatically generated [12].

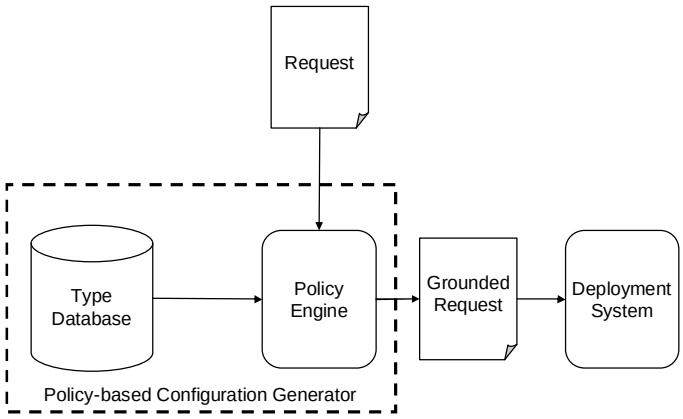


Figure 2: Resource construction process

The composition tool [8] in Quartermaster uses a constraint satisfaction engine [11] to automatically create a design from minimally-specific policies provided by the user. The tool is exposed to the user as a “drag-and-drop” graphical user interface (GUI), which can be used to design complex environments "on-the-fly" from components such as software, firewalls, servers, load balancers, and storage devices. The tool allows operators to define models for a broad range of resources such as web servers, application servers, databases, or even complete e-commerce sites, and provide them to users as resource templates on the GUI. The user can then select components from the palette and design the desired environment. For example, the user can simply drag in the icon representing an e-commerce site into the design panel, add policies representing specific requirements (e.g., the number of transactions per second the site must be capable of supporting, the size of the database, etc.), and ask the tool to complete the design. The composition tool then automatically creates a system design that complies with both the policies specified by the user, as well as those specified as part of the resource templates by the operator. If a design

compliant with the specified policies is not possible, the tool identifies which policies are causing the conflicts.

Figure 2 shows the high level structure of the policy-based composition tool. The user creates a request (which may be minimally specific) for a composed resource. The composition tool uses the Quartermaster type repository and depending on the policies specified in the resource request and those associated with the resource types, generates a "grounded" request specification (i.e., a specification that is provably compliant with policy). The grounded request contains enough detail to allow a deployment system [9] to instantiate the request. The policy engine treats the user's request and the corresponding policy constraints as a goal to be achieved. It uses a constraint satisfaction engine [11] to select resource types and configuration activities, and assigns attribute values such that all of the policy constraints are satisfied.

2.2. Proactive Capacity Management

Once the design is complete, the user submits the design to the capacity manager [13]. Managing capacity of large resource pools in enterprise environments is a challenging problem because, unlike batch environments, applications in enterprise environments are long running but have time-varying resource demands. The capacity manager ensures that sufficient capacity is available to support the longer term aggregate demands of applications.

Quartermaster includes scheduling and capacity management algorithms [14], [15], [16] that can take such fluctuations into account. Complex patterns of time varying resource demands along with resource access Quality of Service requirements can be tracked within Quartermaster and resources can be scheduled accordingly. If necessary, the scheduler may re-schedule low priority requests to accommodate higher priority requests [13].

The capacity management tools in Quartermaster are integrated using a Resource Access Management (RAM) framework. The framework relies on historical traces of application demand to forecast future resource requirements. Applications with irregular or unpredictable demands may need to be provisioned for peak loads and/or rely on the capacity management system's ability to provide best effort access to resources. If no historical information is available, the application can be initially provisioned for its anticipated peak demand, and its behavior can be characterized over time. The framework assumes that each application acquires and releases units of resource capacity as necessary to satisfy the Quality of Service (QoS) requirements of users in an application specific manner. Thus the RAM framework does not attempt to manage the quality of service as seen by the users, but only ensures resource-level Service Level Agreements (SLAs) defined when the application is admitted.

As shown in Figure 3, the RAM framework includes several components: admission control, capacity management, resource allocation, policing, arbitration, and assignment. These components address the following questions: which applications should be admitted (i.e., permitted to make resource reservations), how much resource capacity must be set aside to meet their needs, which applications are currently entitled to resource capacity on-demand, which requests for resource capacity will be satisfied, and which units of resource capacity should be assigned to each application. Answering such questions requires statements of expected resource supply, application demands, and required qualities of service.

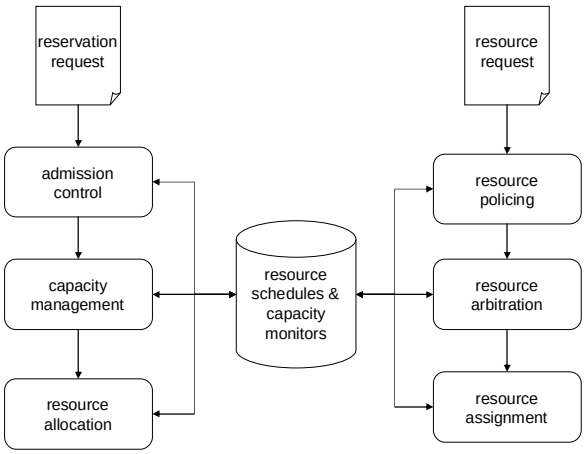


Figure 3: Resource Allocation Framework

Once an application is admitted (i.e., its reservation is accepted), the system and the application have effectively engaged in a contract. The terms of the contract, for example time-varying constraints on resource usage, policy constraints, QoS requirements, pricing, penalties etc., are captured in an SLA. Once admitted, an application must interact with the system to acquire and release resource capacity as needed to meet the QoS requirements of its own users. We assume that the application releases capacity to reduce its own (effective) costs. When an application needs capacity, the system must provide it according to the agreed upon resource-level SLA.

Admission control and resource acquisition processes are illustrated on the left in Figure 3. Admission control decides whether an application will be accepted. It relies on the resource allocation system to determine which resource pools have sufficient capacity to satisfy the demands of the application. Once specific resource pools are chosen, reservations are made and reflected in the capacity management plan. Requests for resource capacity from admitted applications are illustrated on the right in Figure 3. Resource capacity requests are batched by the RAM framework so that tradeoffs can be made regarding which requests for resource capacity are satisfied. Policing mechanisms verify that an application's request for capacity is within the bounds of its SLA. If it is, then the request is entitled to the capacity. If demand exceeds supply, then arbitration mechanisms are used to decide which requests are satisfied. Requests that are to be satisfied are assigned resources. This is discussed in more detail in the next Section.

2.3. Optimized Resource Assignment

If capacity is available, the capacity manager works with a resource assignment system [17], [18] to assign the actual instances of resources for the application. Manual (or simple heuristic) approaches to resource allocation work when all resources are equivalent (e.g., in a cluster), or when the resource pool is small. With complex topologies, it is easy to create bottlenecks in the shared resources when using such approaches, resulting in failure to meet application requirements even when capacity is available in the data center. Quartermaster uses mathematical programming techniques to ensure that bottlenecks are not created in such shared infrastructure.

We refer to the decision making process of choosing the right set of physical resources for each application as application placement, and define the application placement problem (APP) as follows. Given: (1) a topology of a data center consisting of switches, servers, and storage devices with varying capabilities; and (2) a component-based distributed application with requirements for processing, communication and storage; decide which server from the data center fabric should be assigned to each application component. Quartermaster uses a solver based on a mixed integer programming (MIP) formulation to automate application placement on data center fabrics.

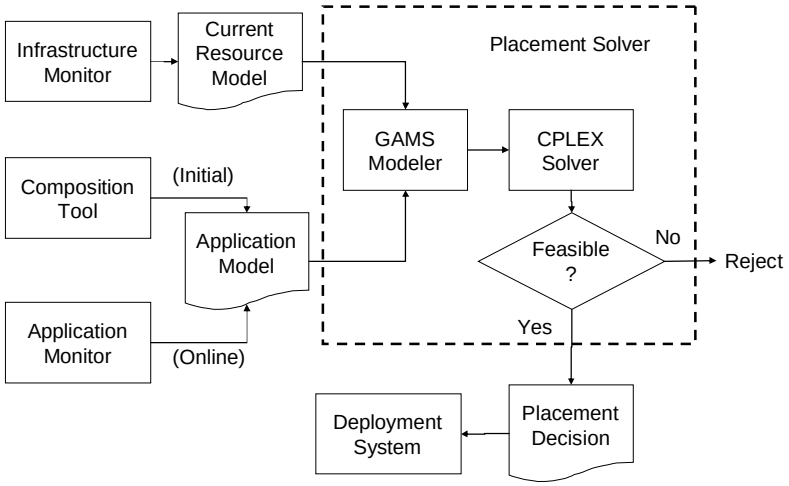


Figure 4: Architecture of the resource assignment solver

Figure 4 shows the architecture of the resource assignment solver. The solver requires two models as input: a resource model and an application model. The resource model describes the fabric topology and the resource capacities. The application model defines the application topology and its resource requirements. The infrastructure monitor tracks the resource inventory, including the connection topology and available capacity. The monitor maintains an up-to-date model of the current state of the environment using information from the resource inventory

and monitoring tools. The application designer uses the composition tool (described earlier) to map the application's high-level QoS goals into an initial application model that represents the low-level processing, communication and storage requirements on the physical resources. The constraints and the objective function required by the solver are dynamically generated from these parameters using the modeling language GAMS [19], and fed into the CPLEX solver [20]. The latter checks the feasibility of the problem, and finds the optimal solution among all feasible solutions.

We have also used the resource assignment system in scenarios that have not included the capacity management methods of the previous section. For example, the application designer uses the composition tool (described earlier) to map the application's high-level QoS goals into an initial application model that represents peak processing, communication and storage requirements on the physical resources. In this scenario, the placement solver is used directly for admission control. If the placement solver decides that no feasible combination of resources can meet the need of the application, the application is denied service or asked to postpone its request. Once the application starts execution in the utility, the application monitor takes over and measures the application's usage of resources, such as network bandwidth, in real-time and updates the application model. The placement solver may be called again if necessary to rebalance load in the utility, to reduce resource fragmentation, to remove hot spots, or to handle infrastructure failures.

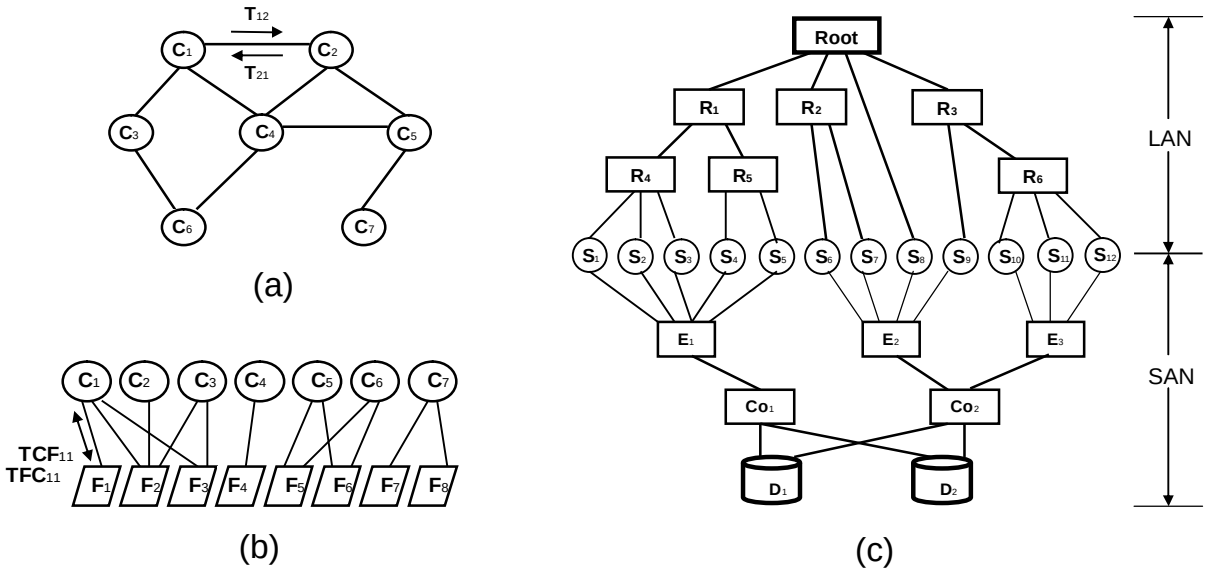


Figure 5: Models used by placement solver: a) application as a set of components with communication requirements, b) storage needs of the application as a set of files, and c) the assumed data center topology

Figure 5(a) and (b) show the structure of the application model. The application is represented as a directed graph, where each of the nodes C_i represents an application component, with requirements on processing and/or storage. Each link in the graph represents the communication requirements T_{ij} between the corresponding components. Note that the placement solver ensures that both networking and processing requirements of the application are met when assigning resources to the application. Storage required for the application is similarly modeled by a set of "files" $\{F_i\}$. Here we use the abstract notion of a file to represent a logically contiguous chunk of data that may be accessed by application components. The storage access pattern is represented by a bipartite graph as shown in the figure. This model can also be used for simultaneous placement of multiple applications by constructing a single big graph with the components from all the applications, where each application is represented by a sub-graph.

The resource model assumed by the placement solver is shown in Figure 5(c). The model represents a typical data center fabric with servers S_i connected over an Ethernet local area network (LAN) fabric consisting of a set of switches R_i, and storage arrays D_i connected to processing elements using a storage area network (SAN) fabric consisting of SAN switches E_i and Co_i. Note that network attached storage (NAS) devices can be modeled as servers in this model, as can other appliances such as firewalls, load balancers, VPN devices and the like. Every device is described by a set of attributes (e.g., processor architecture, CPU speed, memory size, link bandwidth etc.).

By varying the inputs to the solver within the application and data center models, the Quartermaster application placement solver can handle a number of different deployment scenarios:

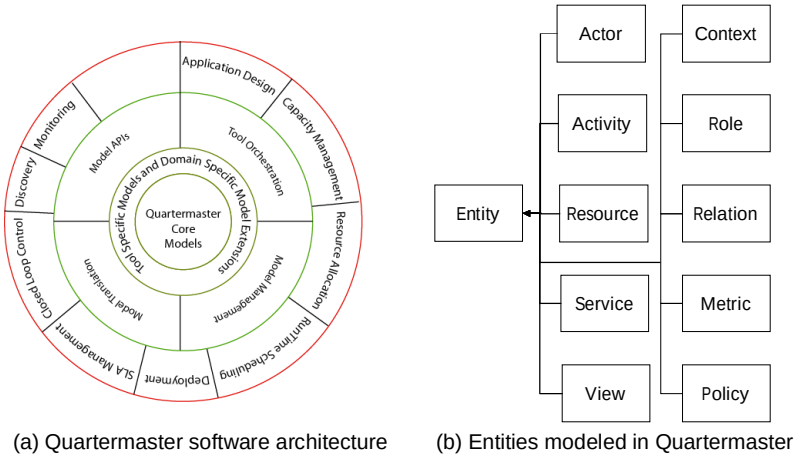
- Greenfield placement: Placing the first application in an empty utility.
- Sequential placement: Placing a new application in an already populated utility.
- Parallel placement: Placing multiple applications in an empty or populated utility at the same time.
- Component migration: Migrating existing application components to avoid hot spots or make more efficient use of resources when the monitoring system detects a problem.
- Automatic fail-over: Finding the best replacement for the failed resources from the free pool.
- Dynamic resource flexing: Providing additional resources to, or taking resources from an existing application, when a new application model is submitted with updated requirements (e.g., as the application's workload changes). Depending on the application's ability to accommodate server migration, our solver can find the new placement solution with or without fixing some subset of the deployed application components.

The first three scenarios happen when an application is deployed, and usually occur at time scales of days or longer. The last three scenarios occur once an application is operational, typically at time scales of minutes to hours.

Note that this approach does not preclude solution of problems where the input data is uncertain. By providing appropriately conservative input requirements, solutions can be found that reflect the known requirements, rather than indiscriminate over-provisioning of infrastructure. In addition, the mathematical programming model is flexible enough to allow a choice of different objective functions or inclusion of other constraints based on customer or operator policies. Thus the approach is valuable as a framework for solving resource assignment problems even when only limited models of the applications and the infrastructure are available.

3. Quartermaster Software Architecture

All tools within Quartermaster are integrated through a repository that maintains both the resource type definitions, as well as the inventory of resources controlled by Quartermaster. All resource types in Quartermaster are modeled using the CIM (Common Information Model) meta-model [21]. CIM is an industry standard defined by the Distributed Management Task Force (DMTF) [22]. By leveraging the CIM meta-model, our system can easily incorporate both existing CIM classes defined for resources, as well as vendor- or system-specific extensions that are required for modeling resources that are currently not defined in CIM. Furthermore, by using a model-driven system, we have found it easy to create "adapters" that can represent those models in a variety of languages (e.g., RSL [23] for the Grid, or the SmartFrog Language [9]).



(a) Quartermaster software architecture (b) Entities modeled in Quartermaster

Figure 6: Software architecture for Quartermaster and entities modeled by it

Figure 6 (a) shows the software architecture for Quartermaster. The type and instance repositories are maintained

within a CIM Object Manager (CIMOM) that has been implemented using a relational database (MySQL). The core entities modeled within Quartermaster are shown in Figure 6 (b). Note that while Quartermaster follows the CIM meta-model, it includes concepts not currently present in the CIM models, and thus extends the CIM models for these entities.

To enable the integration of additional tools, Quartermaster communicates with tools in a loosely-coupled manner. The tools expect information (and provide output) in tool-specific formats. Each tool is integrated using a tool manager, which uses an output adapter to convert model information from the Quartermaster repository to the format required by the tool, and converts the tool output using an input adapter to a form appropriate for the model repository. The tool manager acts as a simple pass-through for information in simple cases, but can have tool-specific logic to enable more complex interactions with the tools. An example of this software pattern is shown in Figure 7, where the interaction with the resource assignment system is shown. The placement tool manager (the Placement Designer) is asked to place an entity (the grounded request). It first uses the output adapter to retrieve all information required for the placement (see Section 2.5) from the repository and converts it into an XML format required by the placement tool. It then uses an HTTP POST operation to send the information through a web server to the CPLEX solver. The assignment information is received by the Placement Designer in an XML format, and the designer converts that information using the input adapter to make the appropriate associations in the repository.

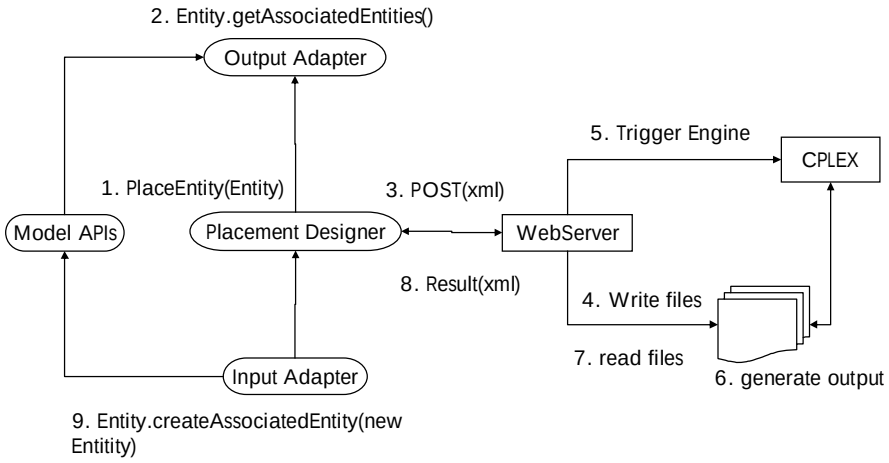


Figure 7: Interaction with resource assignment tool

By using this pattern, many different tools can be incorporated within the Quartermaster tool set, and managed and maintained independently. Furthermore, translations between the model representations and tool specific formats are localized in the adapter classes; hence tool-specific code in Quartermaster is localized. Finally, integration becomes easier since the individual tools do not have to know about the models maintained in the repository, or be reconciled with other tools that are using the same repository.

4. Experience using Quartermaster

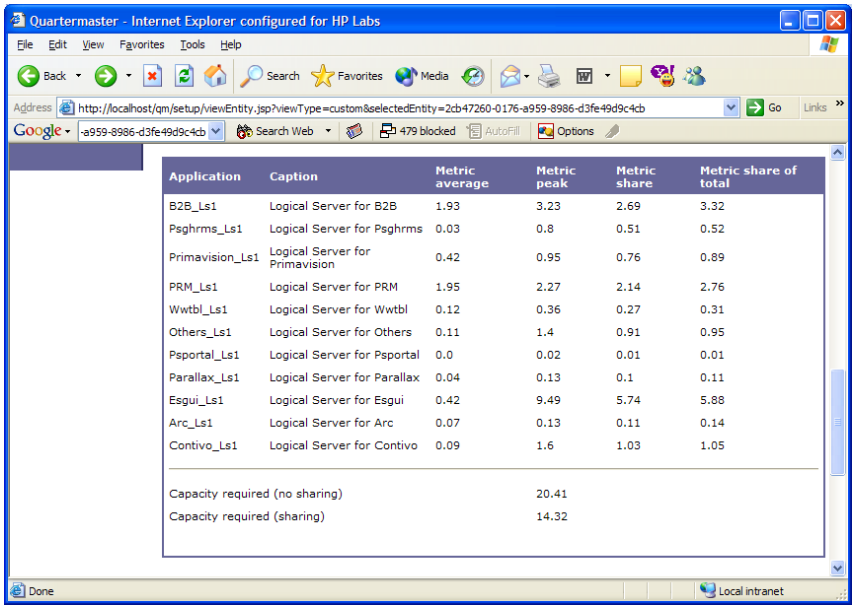
We have integrated the components within Quartermaster into a complete test-bed that we are using to further explore the component algorithms in our research. We have also implemented a prototype that can be used by others to develop additional capabilities. We have used Quartermaster in a number of initiatives to understand its capabilities within service deployment, configuration, and lifecycle management scenarios. We briefly mention two of these initiatives below, as examples of problems that may be tackled by Quartermaster technologies.

HP's Shared Application Server Utility (SASU): Like many large companies, HP maintains and operates hundreds of internal business applications. Each of these applications has traditionally required its own server. HP IT is currently consolidating its J2EE servers into a shared utility that will support the needs of J2EE applications, and provide the application server platform as a utility to groups within HP. This would reduce the licensing, support, management, and hardware costs associated with these applications.

The service is hosted on clusters of HP-UX servers and exploits HP-UX workload management features [29] to

provide performance isolation among services. The capacity management components of Quartermaster are being used as part of this process. The Quartermaster capacity manager supports admission control exercises, recommends which server(s) are best suited for supporting a new application, indicates which services should share servers, and guides the setting of configuration parameters needed for the workload manager that controls the fine grain assignment of server CPU shares to the services.

Figure 8 shows an example output from the capacity management tool. A number of applications are shown which are running on a large HP-UX server. The capacity management tool uses historical traces of the metric (in this case, CPU required) to provide estimates of the number of CPUs required to meet the demands of the applications. As shown at the bottom of Figure 8, for this particular example, it is estimated that if the applications are provisioned without regards to CPU sharing, 21 CPUs will be required, while the sharing of CPUs between applications would allow the capacity needs of the applications to be met with 15 CPUs.



Application	Caption	Metric average	Metric peak	Metric share	Metric share of total
B2B_Ls1	Logical Server for B2B	1.93	3.23	2.69	3.32
Psghrms_Ls1	Logical Server for Psghrms	0.03	0.8	0.51	0.52
Primavision_Ls1	Logical Server for Primavision	0.42	0.95	0.76	0.89
PRM_Ls1	Logical Server for PRM	1.95	2.27	2.14	2.76
Wwtbl_Ls1	Logical Server for Wwtbl	0.12	0.36	0.27	0.31
Others_Ls1	Logical Server for Others	0.11	1.4	0.91	0.95
Psportal_Ls1	Logical Server for Psportal	0.0	0.02	0.01	0.01
Parallax_Ls1	Logical Server for Parallax	0.04	0.13	0.1	0.11
Esgui_Ls1	Logical Server for Esgui	0.42	9.49	5.74	5.88
Arc_Ls1	Logical Server for Arc	0.07	0.13	0.11	0.14
Contivo_Ls1	Logical Server for Contivo	0.09	1.6	1.03	1.05
Capacity required (no sharing)			20.41		
Capacity required (sharing)			14.32		

Figure 8: Typical output of capacity management tool for SASU

Financial Services IT Consolidation Initiative: We are currently working with one of HP's large financial services customers on an IT consolidation initiative. As part of this initiative, the financial services business intends to migrate users to terminal-based "virtualized" desktops with all users served by centralized compute clusters and storage facilities. For testing purposes, we have replicated this environment within our lab using bladed servers. We are using Quartermaster tools within this test environment to create an integrated model-based view of the entire system, including the resources, services, and clients using the environment. Using these models, we plan to explore how cluster resources can be automatically configured and scheduled for these clients based on client requirements for services and the capacity available within the cluster.

Figure 9 shows response times measured on our tested for various tasks for the virtualized desktop. Corresponding capacity demand data is incorporated within the desktop resource models within the Quartermaster tools used for creating desktop configurations for the users, and to both allocate and assign resources to the various desktops.

In both case studies, feedback from the system operators has been positive. In particular, operators have found that the models within Quartermaster help provide an integrated view of the system that is otherwise not available, and focus attention on data that is at the right level of abstraction. In addition, they have found the framework useful because it enables them to quantify improvements (e.g., utilized capacity or time-to-deployment of applications) that are otherwise difficult to evaluate.

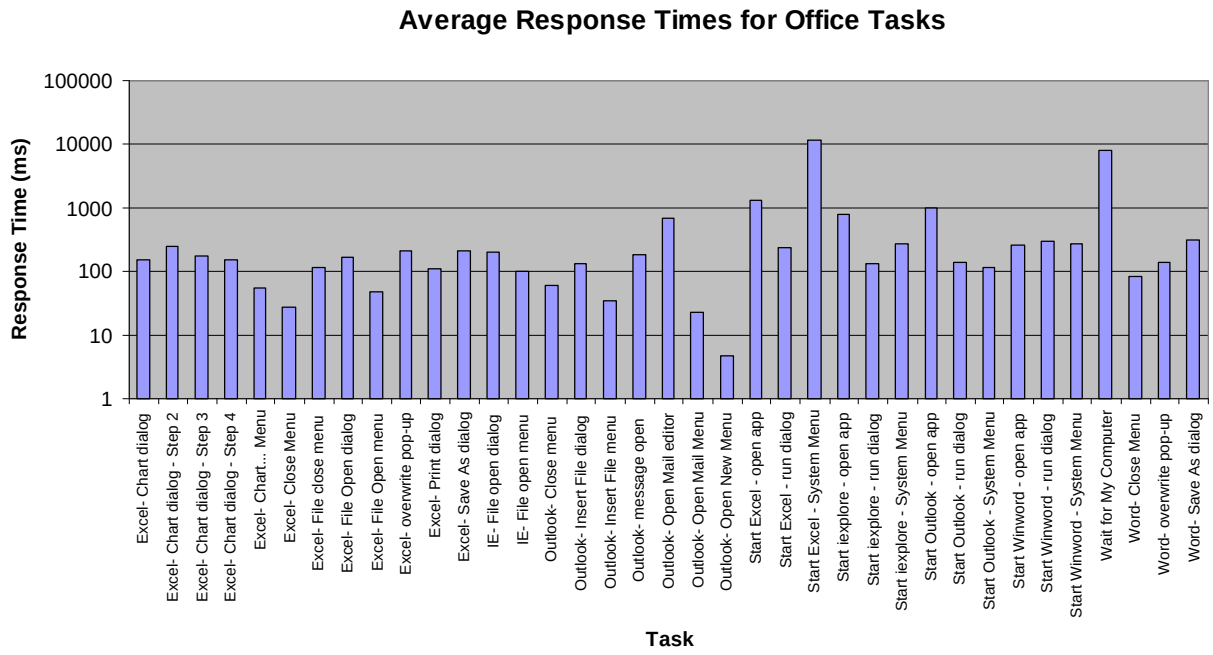


Figure 9: Typical time observed for various office tasks in a virtualized desktop

We are currently exploring within these case studies how custom views can be created for operators. In addition, based on user feedback, we plan to include other capabilities that would provide Quartermaster with:

- the ability to measure the behavior of the designed system at run-time and automatically adapt the design to maintain it within user-specified bounds;
- measurements of application behavior to enable iterative improvement and refinement of the component types; and
- real-time data from the resource pool to our resource allocation and placement algorithms, so that inaccuracies in the parameters specified in the design do not accumulate in practice.

5. Related Work

Most of the individual problems tackled within Quartermaster have been described in the literature. System composition has been explored within the artificial intelligence community [24]. A rich literature exists on problems of scheduling resources to applications in the computer science community, as well as in the high performance computing community [25]. Similarly, the statistics [26] and operations research communities [27] have studied algorithms for modeling time varying quantities and optimization algorithms respectively. Finally, system modeling has been subject to much research within the software engineering field [28]. However, Quartermaster brings together knowledge from these many diverse areas to create an end-to-end framework designing and deploying applications.

6. Summary

Quartermaster seeks to create an IT resource utility, where complex applications can be provisioned for IT users on-demand. Achieving this is difficult because the needs of the enterprise users are complex. Each application running within the enterprise has unique assumptions, each enterprise has different policies that are associated with its applications, and each user brings a different set of requirements for their application. Quartermaster takes an end-to-end lifecycle view of applications. It contains an integrated set of tools that provide users with the ability to compose complex environments, manage capacity within resource pools, and allocate resources from those resource pools to applications and users. The individual tools can be used either by themselves, or in combination with other

available tools and technologies to tackle a host of related problems in system design, planning, and operational environments.

For example:

- The composition tool can be used by system designers as an application design aid for maintaining and upgrading their designs and capturing "best-practices" as policies to ensure that designs conform to those policies.
- The capacity manager can be used for tracking and scheduling resources between applications in bladed systems or for high-performance computing or applications where statistically varying workloads are present.
- The resource assignment tools can be used within the datacenter environments to ensure that application-level requirements are met without creating resource bottlenecks when applications are deployed.

We are using these tools in a number of initiatives both within HP and with HP customers to test their applicability within different use cases, to obtain feedback and experience from their use, and to refine the tools using this experience. The Quartermaster architecture provides us a framework within which these (and other) capabilities are integrated.

7. References

[1] HP Utility Data Center (UDC) http://www.hp.com/solutions1/infrastructure/solutions/utilitydata

[2] IBM Autonomic Computing http://www.ibm.com/autonomic

[3] SUN N1 http://wwws.sun.com/software/solutions/n1/

[4] Sychron http://www.sychron.com

[5] Opsware http://www.opsware.com

[6] VMWare http://www.vmware.com

[7] Troux http://www.troux.com

[8] A. Sahai, S. Singhal, V. Machiraju, R. Joshi, "Automated Policy-Based Resource Construction in Utility Computing Environments," 2004 IEEE/IFIP Network Operations and Management Symposium, Seoul, Korea, April 2004.

[9] SmartFrog http://www.smartfrog.org/

[10] OpenView http://openview.hp.com/

[11] C. Flanagan, R. Joshi , X. Ou,, J. Saxe, "Theorem Proving Using Lazy Proof Explication," In Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Volume 2725, pp. 355-367, Jul 2003.

[12] A. Sahai, S. Singhal, R. Joshi, V. Machiraju, "Automated Generation of Resource Configurations through Policies," IEEE 5th International Workshop on Policies for Distributed Systems and Networks, YorkTown Heights, June 2004.

[13] J. Pruyne and V. Machiraju, "Quartermaster: Grid Services for Data Center Resource Reservation," Global Grid Forum Workshop on Designing and Building Grid Services, October 8, 2003, Chicago, Illinois, USA

[14] J. Rolia, X. Zhu, M. Arlitt and A. Andrzejak, "Statistical Service Assurance for Applications in Utility Grid Environments," IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, Ft. Worth TX, October 2002.

[15] J. Rolia, X. Zhu and M. Arlitt, "Resource Access Management for a Resource Utility for Commercial Applications," IEEE/IFIP Int. Symposium on Integrated Network Management, Colorado Springs, CO, March 2003.

[16] J. Rolia, A. Anderzejak, and M. Arlitt, “Automating Enterprise Application Placement in Resource Utilities,” IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, LCNS 2867, M. Brunner and A. Keller (eds), pp. 118-129.

[17] X. Zhu and S. Singhal, “Optimal Resource Assignment in Internet Data Centers,”- IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, Cincinnati, OH, August 2001.

[18] X. Zhu, C. Santos, J. Ward, D. Beyer, S. Singhal, “Resource Assignment for Large Scale Computing Utilities using Mathematical Programming,” HPL Tech. Rep. HPL-2003-243, November 2003.

[19] GAMS, www.gams.com

[20] CPLEX, www.ilog.com

[21] CIM Modeling http://www.dmtf.org/standards/standard_cim.php

[22] DMTF: http://www.dmtf.org

[23] Globus Resource Specification Language http://www.globus.org/gram/rsl_spec1.html

[24] F. Brazier, C. Jonker, J. Treur, “Principles of Compositional Multi-Agent System Development,” Proc. of the IFIP’98 Conference IT&KNOWS’98, J. Cuena (ed.), Chapman and Hall, 1998

[25] S. Chang, J. A. Stankovic and K. Ramamritham, “Scheduling algorithms for hard real-time systems: a brief survey,” in J. A. Stankovic and K. Ramamritham (eds), Hard Real-Time Systems: Tutorial, IEEE, 1988, pp. 150--173.

[26] S. Levinson, “Statistical modeling and classification,” in Survey of the State of the Art in Human Language Technology, Cambridge University Press, 1996.

[27] C. Harvey, “Operations Research: An Introduction to Linear Optimization and Decision Analysis,” Elsevier Science, 1979

[28] A. Felfernig, G. E. Friedrich et al. UML as a domain specific knowledge for the construction of knowledge based configuration systems. In the Proceedings of SEKE'99 Eleventh International Conference on Software Engineering and Knowledge Engineering, 1999.

[29] HP Workload manager http://h30081.www3.hp.com/products/wlm/