Towards a Secure and Efficient Model for Grid Computing using Mobile Code

Walter Binder[1], Giovanna Di Marzo Serugendo[2], and Jarle Hulaas[2]

[1] CoCo Software Engineering GmbH, Vienna, Austria
[2] Computer Science Department, University of Geneva, Switzerland

Abstract. Mobile code has often been mentioned as an attractive technology for distributing computations inside a Grid consisting of heterogeneous nodes interconnected by a large-scale network. We describe here a Java-based mobile agent model for a Grid infrastructure which addresses issues such as customizable distribution of computation, security, billing and accounting.
Keywords: Mobile Agents, Resource Control, Grid Computing.

1 Introduction

Grid computing enables worldwide distributed computations involving multi-site collaboration, in order to benefit from their combined computing and storage power. The way to distribute an application on a set of computers connected by a network depends on several factors.

First, it depends on the *application* itself, which may be not naturally distributed or on the contrary may have been engineered for Grid computing. A single run of the application may require a lot of computing power. The application is intended to run several times on different input *data*, or few times, but an a huge amount of data. The application has at its disposal computational, storage and network *resources*. They form a dynamic set of CPUs of different computing power, of memory stores (RAM and disks) of different sizes, and of bandwidths of different capacities. In addition, the basic characteristics of the available CPUs, memory stores and bandwidth are not granted during the whole computation (a disk with an initial capacity of 512MBytes when empty, cannot be considered having this capacity when it is half full). Code and data may be stored at different *locations*, and may be distributed across several databases. Computation itself may occur at one or more locations. Results of the computation have to be collected and combined into a coherent output, before being delivered to the client, who may wait for it at still another location. The network *topology* has also an influence on the feasibility of the distribution. Centralized topologies offer data consistency and coherence by centralizing the data at one place, security is more easily achieved since one host needs to be protected. However, these systems are exposed to lack of extensibility and to fault-tolerance, due to the concentration of data and code to one location. On the contrary, a fully decentralized system will be easily extensible and fault-tolerant, but security and data coherence will be more difficult to achieve. A hybrid approach combining a set of servers, centralizing each several peers, but organized themselves in a decentralized network, provides the advantages of both topologies [17]. Finally, *policies* have to be taken into account. They include clients and donators (providers) requirements, access control, accounting, and resource reservations.

Mobile agents constitute an appealing concept for realizing computation distributions, since the responsibility for dispatching the program or for managing some run-time tasks may be more efficiently performed by a mobile entity that rapidly places itself at strategic locations. However, relying completely on mobile agents for realizing the distribution complicates security tasks, and may incur additional network traffic.

This paper proposes a model combining the use of a stationary operator with mobile agents, running in a secure Java-based mobile agent kernel. The operator is responsible for centralizing customized clients computations requests, as well as security and billing tasks, and for dispatching the code on the Grid. Mobile agents prevent the operator to become a bottleneck, by forwarding input data to computations locations, and performing management tasks. They start the different parts of the computations, ensure the management and monitoring of the computation's distribution, and finally collect and combine the results of the computation.

Section 2 reviews distributed computations, Section 3 presents the model, Section 4 describes the chosen secure execution environment, while Section 5 explains adaptations of the environment necessary to the full realization of the model. Finally, Section 6 summarizes some related approaches.

2 Distributed Computations

Worldwide distributed computations range from parallelization of applications to more general Grid distributions.

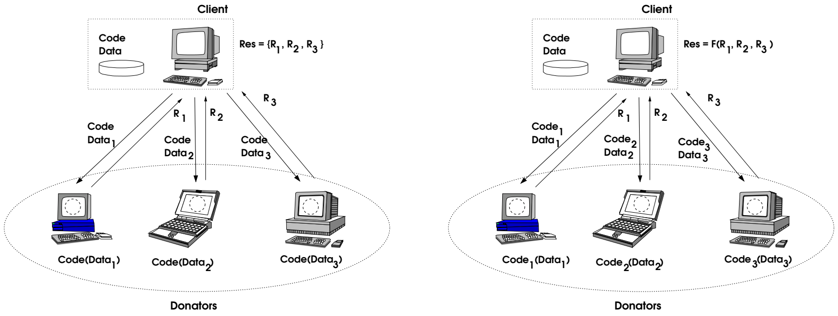
2.1 Parallelization

Distribution of computing across multiple environments shares similarities with the parallelization of code on a multi-processor computer. We distinguish two cases, the first one corresponds to single instruction, multiple data (SIMD); while the second one corresponds to multiple instruction, multiple data (MIMD). Figure 1 shows both cases.

In case (a), the client's host ships the same code, but with a different accompanying data to multiple locations. After computation, the different results are sent back to the client's host. The final result is simply the collection of the different results. This kind of distribution is appropriate for intensive computing on a huge amount of the same type of data. It corresponds to the distribution realized by the SETI@home[3] experiment that uses Internet connected computers in the Search for Extraterrestrial Intelligence (SETI). Donators first download a free program. The execution of the program then downloads and analyzes radio telescope data. Note that in this case, the downloaded data may come from a different source.

In case (b), code and data are split into several parts, then pairs of code and data are sent to several locations. The result is obtained by a combination (some function) of the different results.

[3] http://setiathome.ssl.berkeley.edu/



(a) SIMD (b) MIMD

Fig. 1. Parallelisation

Such a distribution is suitable for applications that can be divided into several pieces. This scheme fits the case of Parabon[4]. The client defines jobs to be performed. Transparently, the API divides the job into several tasks, on the client side; a task is made of a code, data, and some control messages. Tasks are sent to the Parabon server, which then forwards each task to a donator, using the donator's CPU idle time for computing the task. Once the task is achieved, the server sends back the result to the client, where the API then combines all results together, before presenting them to the client.

As a particular case of the MIMD example, code and data may be divided into several sequential parts. Computation would occur then in a pipeline-like style, where the next piece of code runs on the result of the previous computation.

These examples all exploit idle CPU time of the computer participating in the computations. The execution of the code on the data occurs inside a secure "bubble", which ensures, on one hand, that the donator cannot exploit the code, the data and the results of the client; on the other hand, that the client does not execute malicious code in the donator's host.

2.2 Grid

The more general case of distributed computing is provided by the Grid computing concept which enables collaborative multi-site computation [11]. Grid computing goes beyond traditional examples of peer-to-peer computing, since there is a concern of proposing a shared infrastructure for direct access to storage and computing resources.

Figure 2 shows a generic Grid computation, encompassing the different classes of Grid applications [10]. The client, requesting the computation, the software to run, the data, and the results may be located at different sites. The data is even distributed across two databases. In this example, the code and the two pieces of

[4] http://www.parabon.com

data are moved to the donator's location, where the computation takes place. The result is then shipped to the client.

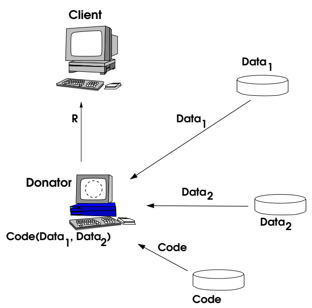


Fig. 2. Grid

The CERN DataGrid [8] provides an example where physicists are geographically dispersed, and the huge amount of data they want to analyze are located worldwide.

3 Proposed Model

In this section we give an overview of our overall architecture, we outline our business model, and describe the different roles of participants in our Grid computing infrastructure, as well as their interactions.

3.1 Participating Parties in the Grid Computing Model

Our model involves 3 distinct parties: the *operator* of the Grid, *resource donators*, and *clients*. The operator is in charge of maintaining the Grid. With the aid of a mobile *deployment agent*, he coordinates the distribution of applications and of input data, as well as the collection and integration of computed results. The operator downloads the applications, and distributes them to resource donators that perform the actual computation.

Clients wishing to exploit the Grid for their applications have to register at a server of the operator before they are allowed to start computations. During the registration step, the necessary information for billing is transmitted to the operator. Afterwards the client is able to send a *deployment descriptor* to the operator.

The deployment descriptor comprises the necessary information allowing the operator to download the application, to prepare it for billing and accounting, and to distribute the application and its streams of input data to different active resource donators, taking into consideration their current load. The mobile deployment agent, which is created by the operator based on the contents of the client's deployment descriptor and coordinates the distributed client application, is not bound to a server of the operator; the client may specify the server to host the deployment agent, or decide to let the agent roam the Grid according to its own parameters. This approach improves scalability and ensures that the operator does not become a bottleneck, because the operator is able to offload deployment agents from his own computers.

Resource donators are users connected to the same network as the operator (e.g., the Internet) who offer their idle computing resources for the execution of parts of large-scale scientific applications. They may receive small payments for the utilization of their systems, or they may donate resources to certain kinds of applications (e.g., applications that are beneficial for the general public). Resource donators register at the Grid operator, too. They receive a dedicated execution environment to host uploaded applications. Portability, high performance, and security are key requirements for this execution platform. Section 4 gives detailed information on our platform, which is completely based on the Java language. The operator dispatches downloaded applications to active resource donators. The deployment agent is in charge of supervising the flows of initial and intermediate data to and from the resource donators, as well as the final results, which are passed back to the destination designated by the client. Allowing the deployment agent to be moved to any machine on the Grid improves efficiency, as the deployment agent may locally access the required data there. As explained later, the deployment agent, or its clones, is also responsible for minimizing the flows of Grid management data between the donators and the operator.

3.2 Business Model

In our model the operator of the Grid acts as a trusted party, since he is responsible of all billing tasks[5]. On the one hand, clients pay the operator for the distributed execution of their application. On the other hand, the operator pays the resource donators for offering their idle computing resources.

The client buys *execution tickets* (special tokens) from the operator, which the deployment agent passes to the resource donators for their services. The resource donators redeem the received execution tickets at the operator. The execution tickets resemble a sort of currency valid only within the Grid. They are micro-payments for the consumption of computing resources. There are 3 types of execution tickets: tickets for CPU utilization, for memory allocation, and for data transfer over the network. The coordinating deployment agent has to pass execution tickets of all types to a resource donator for exploiting his computing resources.

Execution tickets are protected from faking, as the operator keeps track of the tickets in use. Execution tickets can be split and distributed at a fine granularity. Hence, the loss of a single execution ticket (e.g., due to the crash of a resource donator) is not a significant problem. In case the deployment agent does not receive the desired service from a resource donator for a given execution ticket, it will report to the operator. If it turns out that a resource donator collects tickets without delivering the appropriate service, the operator may decide to remove him from the Grid.

[5] The operator may also be responsible for guaranteeing that only applications corresponding to the legal or moral standards fixed by the donators are deployed.

3.3 Deployment of Applications

In order to start an application, the client transmits a deployment descriptor to the operator, who will retrieve and dispatch the application to different resource donators and also create a deployment agent for the coordination of the distributed execution of the application.

The deployment descriptor, sent by the client, consists of the following elements:

- A description of the application's code location and structure. The client informs the operator of the application he wants to run. The operator will then download the application, and prepare it for resource control, before dispatching it to the donators. The application's structure establishes cut points and defines the different parts of the application that can run concurrently, as well as possible computational sequences. The client may specify himself the composition of computations, which reflects the calculus he desires to achieve (SIMD, MIMD, other). However, he does not customize the part of the description related to cut points, since it is tightly dependent of the application;
- A description of the source for input data. Usually, scientific applications have to process large streams of input data, which can be accessed e.g. from a web service provided by the client. The interface of this service is predefined by the operator, it may support various communication protocols (e.g., RMI, CORBA, SOAP, etc.);
- A descriptor of the destination for output results. Again, this element designates the location of an appropriate service that can receive the results;
- Quality-of-service (QoS) parameters. The client may indicate the priority of the application, the desired execution rate, the number of redundant computations for each element of input data (to ensure the correctness of results), whether results have to be collected in-order or may be forwarded out-of-order to the client, etc. The QoS parameters allow the client to select an appropriate trade-off between execution performance, costs, and reliability. The QoS parameters are essential to select the algorithms to be used by the deployment agent. For instance, if the client wishes in-order results, the deployment agent may have to buffer result data, in order to ensure the correct order.

In the following we summarize the various steps required to deploy a client application in the Grid. Figure 3 illustrates some of them.

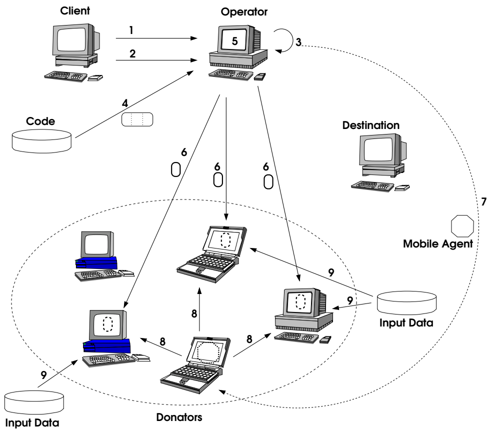
1. Prospective resource donators and clients download and install the mobile code environment employed by the chosen operator, in order to be able to run the computations and/or to allow the execution of deployment agents.
2. Donators register with the operator and periodically renew their registration by telling how much they are willing to give in the immediate future; a calibration phase is initially run at each donator site to determine the local configuration (processor speed, etc.).
3. A client registers with the operator and sends the deployment descriptor (steps 1 and 2 of Figure 3).
4. The operator reads the deployment descriptor and:

(a) Chooses an appropriate set of donators according to the required service level and to actually available resources; a micro-payment scheme is initiated, where fictive money is generated by the operator and will serve as authorization tokens for the client to ask donators for resources; a first wave of credit is transferred to the donator set, thus signifying that the corresponding amount of resources are reserved.
(b) Creates a mobile agent, the deployment agent, for coordinating the distribution, execution and termination of the client application (step 3); this deployment agent will shift the corresponding load from the operator to the place designated by the client, or to a donator chosen according to load balancing principles; the deployment agent may clone itself or move to the appropriate places for ensuring that input and output data is transferred optimally, thus avoiding useless bottlenecks at central places like the operator server.
(c) Downloads the client application (step 4) and rewrites it (reification of resources, step 5); the resulting code is signed to prevent tampering with it, and deployed directly from the operator's server (step 6).
(d) Dispatches the deployment agent to the appropriate initial place for execution (step 7).

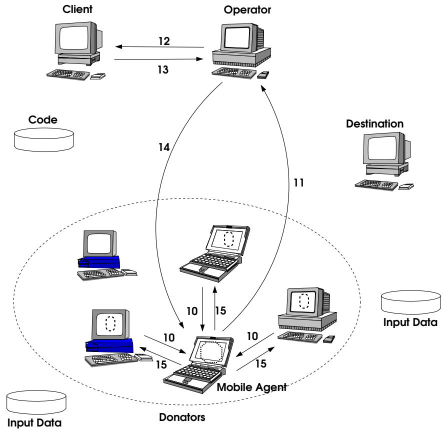
5. The deployment agent launches the distributed computation by indicating (step 8) to each donator-side task where to locate its respective share of input data (step 9), and starts monitoring the computation.
6. The deployment agent acts as a relay between the operator and the donators. The agent receives regular status reports from the various locations of the resource-reified application (step 10); this enables him to monitor the progress of the computations, and to detect problems like crashes and to assist in the recovery (e.g. by preparing a fresh copy of the appropriate input data, or by finding a new donator to take over the corresponding task); the status reports are filtered and forwarded to the operator (step 11) in order to help maintaining a reasonably good view of the global situation (the operator might decide to schedule a second application on under-utilized donators); when necessary, the operator will ask the client for more credit (step 12), who will then buy more authorization tokens from the operator (step 13). The deployment agent then passes the execution tickets to the donators (steps 14 and 15)
7. When results have to be collected, the deployment agent may clone or migrate to the destination (step 16) and coordinate the incoming flows of data (step 17). He may perform a last filtering and integrity control of data before it is definitely stored.

We favored this model over a P2P setting, since it simplifies the implementation of a global strategy for load balancing and ensures that a trusted party – the operator – controls the progressing computations. In this setting, the operator also is in a natural position for managing all operations related to the validation of client-side payments and corresponding authorizations. Using mobile code for the deployment agent ensures that the server of the operator does not become a bottleneck and single point of failure. In the current model, the application code has to be transferred to the operator's computer, since it needs to be checked for security purposes, to be prepared for billing and accounting (using resource reification), and to be partitioned according to the deployment descriptor.

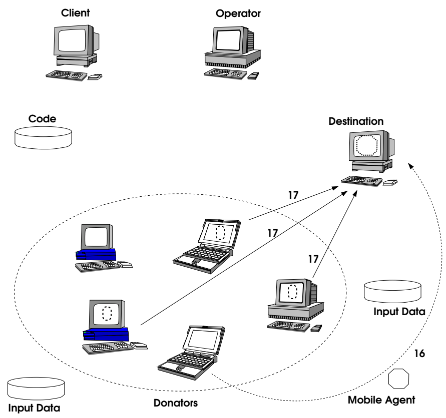
Currently, resource reification is a heavy process, therefore it is natural to concentrate it at the operator's site. However, future work on the resource reification may allow schemes where the application code is directly dispatched to the donators without passing through the operator. Resource reification would then occur at the donators sites.



(a) Deployment



(b) Monitoring



(c) Results

Fig. 3. Application Distribution

4 Using Java for the Distribution of Computations

Here we motivate the use of Java [13] for the implementation of distributed computations and their distribution within a network. In our model we use Java-based mobile agents for the distribution of deployment agents (to a server specified by the client) and of computational tasks (to resource donators). A secure mobile agent kernel, the J-SEAL2 system, serves as execution platform for mobile agents in both cases. In that way, we leverage the benefits of Java and of mobile code, while at the same time offering enhanced security to protect hosts from faulty applications.

4.1 Why Java?

Recently, platforms for Grid computing have emerged that are implemented in Java. For instance, Parabon offers an infrastructure for Grid computing which is based completely on Java. In fact, the Java language offers several features that ease the development and deployment of a software environment for Grid computing. Its network-centric approach and its built-in support for mobile code enable the distribution of computational tasks to different computer platforms.

Java runtime systems are available for most hardware platforms and operating systems. Because of the heterogeneity of the hardware and of operating systems employed by Internet users, it is crucial that a platform for large-scale Grid computing be available for a large variety of different computer systems. Consequently, a Java-based platform potentially allows every computer in the Internet to be exploited for distributed, large-scale computations, while at the same time the maintenance costs for the platform are minimal ("write once, run everywhere").

Apart from its portability and compatibility, language safety and a sophisticated security model with flexible access control are further frequently cited advantages of Java. As security is of paramount importance for the acceptance of a platform for Grid computing, the security and safety features of Java are highly appreciated in this context.

4.2 Performance Issues

Java has its origins in the development of portable Internet applications. The first implementations of Java runtime systems were interpreters that inefficiently executed Java Virtual Machine (JVM) bytecode [16] on client machines. Also, several features of the Java programming language impact performance: the fact that it is a type safe, object-oriented, general-purpose programming language, with automatic memory management, and that its implementation does not directly support arrays of rank greater than one, means that its execution may be less efficient compared to more primitive or specialized languages like C and Fortran.

However, optimizations performed by current state-of-the-art Java runtime systems include the removal of array bounds checking, efficient runtime type checking, method inlining, improved register allocation, and the removal of unnecessary synchronization code. See [15] for a recent survey of current compilation and optimization techniques that may boost the performance of Java runtime systems for scientific computing. In [18] the authors report that some Java applications already achieve 90% of the performance of equivalent compiled Fortran programs.

Considering the advantages of Java for the development and deployment of platforms for Grid computing, we think that a minor loss of performance can be accepted. Furthermore, the availability of more nodes where distributed computations can be carried out may often outweigh minor performance losses on each node. Ultimately, we are confident that maturing Java runtime systems will offer continuous performance improvements in the future.

4.3 Security Considerations

A high level of security is crucial for the acceptance of a platform for Grid computing. At first glance, Java runtime systems seem to offer comprehensive security features that meet the requirements of an execution environment for Grid computing: language safety [24], classloader namespaces and access control based on dynamic stack introspection. Despite these advantages, current Java runtime systems are not able to protect the host from faulty (i.e. malicious or simply bugged) applications.

In the following we point out serious deficiencies of Java that may be exploited by malicious code to compromise the security and integrity of the platform (for further details, see [6]). Above all, Java is lacking a *task model* that could be used to completely isolate software components (applications and system services) from each other. A related problem is that, unfortunately, thread termination in Java is an inherently unsafe operation, which may e.g. leave shared objects, such as certain internals of the JVM, in an inconsistent state. Also related to the lack of task model is the absence of *accounting and control of resource consumption* (including but not limited to memory, CPU, threads, and network bandwidth). Concerning the implementation of current standard Java implementations, an issue is that several bytecode verifiers sometimes accept bytecode that does not represent a valid Java program: the result of the execution of such bytecode is undefined, and it may even compromise the integrity of the Java runtime system. Finally, while the security model of Java offers great flexibility in terms of implementing access control, it lacks central control: security checks are scattered throughout the classes, and it is next to impossible to determine with certainty whether a given application actually enforces a particular security policy.

All these shortcomings have to be considered in the design and implementation of Java-based platforms for Grid computing. Therefore, massive re-engineering efforts are needed to create sufficiently secure and reliable platforms.

4.4 A Java Micro-Kernel for the Secure Execution of Mobile Code

Here we present J-SEAL2 [3, 4], a lightweight security layer that executes on top of standard Java runtime systems. It provides solutions to all the security problems mentioned before and, hence, represents a state-of-the-art platform for the creation of secure environments for Grid computing. Several researchers have stressed the importance of multi-tasking features for Java-based middleware [2]. An abstraction similar to the *process* concept in operating systems is necessary in order to create secure execution environments for mobile code. However, proposed solutions were either incomplete or required modifications of the Java runtime system. In contrast, the J-SEAL2 kernel has been designed to ensure important security guarantees without requiring any native code or modifications of the underlying Java implementation.

J-SEAL2 is a micro-kernel implemented in pure Java, which supports the *hierarchical task model* of the Seal Calculus [22] that was first implemented by the JavaSeal mobile object system [7]. The J-SEAL2 kernel manages a tree hierarchy of nested tasks, which may be either mobile objects or system services. In J-SEAL2 tasks are completely separated from each other. Untrusted tasks are not allowed to directly use certain functions of the JDK, such as file or network IO, but they have to access dedicated J-SEAL2 services that are executing in separate tasks. The hierarchical task model allows to interpose supervisor and mediator components between an untrusted application and trusted system services. Each service access is subject to verification by the supervisor task. Consequently, the J-SEAL2 kernel does not rely on the security model of Java, but enables the installation of centralized security policies.

In J-SEAL2 each task has associated its own set of threads, which cannot cross task boundaries arbitrarily. This property ensures that the termination of a task cannot leave a different task in an inconsistent state, because the threads, which have to be stopped during task termination, are confined to their owning task. Mobile objects are not allowed to directly create Java threads, but they have to use a safe wrapper class instead. The J-SEAL2 kernel enforces additional constraints on mobile objects, in order to ensure that a parent task may terminate its children at any time, forcing the children to release all allocated resources immediately.

Many security restrictions are ensured by *extended bytecode verification*: J-SEAL2 employs a custom classloader, which invokes the extended bytecode verifier of J-SEAL2 before a class is linked by the JVM. This mechanism is also used to prevent untrusted code from accessing certain JDK functions, and compensates for security flaws of the basic JVM verifier.

J-SEAL2 supports *resource control* for physical resources (e.g., memory, CPU, network bandwidth, etc.) and for logical resources (e.g., threads, number of tasks, etc.). According to the nested task model, J-SEAL2 supports hierarchical resource control, where the parent task acts as a resource manager of its children [5]. Since currently Java has no support for resource control, J-SEAL2 relies on bytecode rewriting to reify the memory and CPU consumption of applications [21]. Memory allocation instructions are re-directed to a controller object that denies object allocation if a limit is exceeded. CPU accounting is based on the number of executed bytecode instructions. Accounting is performed at the beginning of each basic block of code; the information on CPU consumption is used by a periodic scheduler thread, which assigns thread priorities according to CPU limits and recent CPU consumption. Due to a carefully tuned accounting scheme, the runtime overhead is kept reasonably small [5]. The consumption of network bandwidth is directly controlled by a supervisor task that is interposed in the hierarchy between untrusted applications and the network service. A limit on the number of concurrent tasks or threads is enforced by the kernel before a new task or thread is created.

The J-SEAL2 micro-kernel is thus perfectly suited for the development of platforms for Grid computing: It is small in size (about 100KB of Java class files) and compatible with the Java 2 platform. Therefore, the distribution and installation of the kernel itself incurs only minimal overhead. J-SEAL2 supports mobile objects, which enable the distribution and remote maintenance of scientific applications. The extended bytecode verifier prevents untrusted code from utilizing dangerous JDK functions. Such restrictions may be cumbersome for developing interactive applications, but typical scientific applications do not need high-level JDK functionalities (except for mathematics and cryptography packages). Finally, whereas scientific applications make heavy use of CPU and memory resources, the resource control features of J-SEAL2 ensure a fair distribution of computational resources among multiple applications and prohibit an overloading of the machine.

5 Adapting J-SEAL2 for Grid Computing

J-SEAL2 has been conceived as a kernel for execution environments for mobile objects, the goal being to achieve savings in network bandwidth, to support offline operation, and to enhance flexibility, especially concerning the distribution and remote maintenance of software. J-SEAL2 has an open and extensible system architecture, allowing to plug-in the necessary services without modifying the code of the kernel.

5.1 Components of the Grid Computing Platform

Essentially, five special components are needed for the Grid computing platform run by the resource donators: A mediator component to control the execution of uploaded applications, a network service to receive application code (Net-App service), a second network service allowing applications to receive input data and to transmit their results (Net-Data service), a system monitor to prevent an overloading of the machine, as well as a monitor window that displays information regarding the running applications, the elapsed time, etc. to the resource donator. In the following we give an overview of these components:

– In the task hierarchy, the *mediator component* is the parent of the untrusted applications. The mediator is responsible for the installation and termination of applications, as well as for access and resource control. It utilizes the Net-App service to receive control messages from the deployment agents that coordinate the distributed applications. It receives application archives, which contain the application code as well as a deployment descriptor. The deployment descriptor comprises a unique identifier of the application, as well as information concerning the resource limits and the priority of the application. The unique application identifier is needed for dispatching messages to the appropriate application. Requests to terminate an application are also received from the Net-App service. The mediator component ensures that applications employ only the Net-Data service and guarantees that an application only receives its own input data and that its output data is tagged by the application identifier. The mediator task uses the system monitor in order to detect when the machine is busy; in this case, applications are suspended until the system monitor reports idle resources.

- The *Net-App service* is responsible for exchanging system messages with the coordinating deployment agent. When the platform is started, the Net-App service contacts the operator's server, which may transfer application archives to the platform. Optionally, a *persistency service* can be used to cache the code of applications that shall be executed for a longer period of time. The Net-App service also receives requests to terminate applications that are not needed anymore.
- The *Net-Data service* enables applications to receive input data and to deliver the results of their computation to the coordinating server. Messages are always tagged by an application identifier in order to associate them with an application. Access to the Net-Data service is verified by the mediator component. Frequently, continuous streams of data have to be processed by applications. The Net-Data service supports (limited) buffering of data to ensure that enough input data is available to running applications. The optimized inter-task communication mechanisms of J-SEAL2 [4] help to minimize the overhead of passing data streams over task boundaries.
- The *system monitor* has to detect whether the machine is busy or idle. If the computer is busy, applications shall be suspended in order to avoid an overloading of the machine. If the computer is idle, applications shall be started or resumed. An implementation of the system monitor may employ information provided by the underlying operating system. However, such an approach compromises the full portability of all other components, since it relies on system-dependent information. Therefore, we follow a different approach: The reification of CPU consumption in J-SEAL2 [5, 21] allows to monitor the progress of applications. If the number of executed instructions is low (compared to the capacity of the hosting computer), even though applications are ready to run, the system monitor assumes that the computer is busy. Therefore, it contacts the mediator component in order to suspend computations. Periodically, the system monitor resumes its activity in order to notice idle computing resources. When the computer becomes idle, all applications are resumed.
- The *monitoring window* presents information about the past and current work load of the system to the resource donator. It shows detailed status information of the running applications, the time elapsed for the computations, the estimated time until completion, if available, as well as some general information regarding the purpose of the computation. As the resource donator is in control of his system, it is important to show him detailed information of the utilization of his machine.

The mobile code execution environment for the deployment agents is based on J-SEAL2 as well. But as the deployment agents stems from the operator, a trusted party, the security settings are relaxed. There are a few mandatory services needed by the deployment agent: access to the client web services that provide the input data and consume the output results, as well as network access for the communication with resource donators and the operator. Communication with the resource donators is necessary for the transmission of the application data, while communication with the operator is essential for the implementation of a global strategy for load balancing and for payment issues.

6 Related Work

The primary purpose of mobile code is to distribute applications and services on heterogeneous networks. Many authors relate mobile code, and more often mobile agents as a practical technology for implementing load-balancing in wide-area networks like the Internet. Load-balancing can be either static (with single-hop agents, in the sense that once a task is assigned to a host, it does not move anymore) or dynamic (with multi-hop mobile agents enabling process migration). A recent survey of load-balancing systems with mobile agents is presented in [12]. Security and efficiency have immediately been recognized as crucial by the research community, but it was necessary to wait for technology to mature. Resource monitoring and control is needed for implementing load-balancing, and more generally for realizing secure and efficient systems, but is unavailable in standard Java, and particularly difficult to implement in a portable way. For instance, Sumatra [1] is a distributed resource monitoring system based on a modified JVM called Komodo. See [5] for a further study on the portability of resource monitoring and control systems in Java.

According to [23], almost all Grid resource allocation and scheduling research follows one of two paradigms: centralized omnipotent resource control - which is not a scalable solution - or localized application control, which can lead to unstable resource assignments as “Grid-aware” applications adapt to compete for resources. Our primary goal is however not to pursue research on *G-Commerce* [23], even though we sketch an economical model based on virtual currency. For these reasons, our approach is hybrid. We relax the conservative, centralized resource control model by proposing an intermediary level with our deployment agents, designed to make the architecture more scalable. We have identified a similar notion of mobile coordination agent in [9], with the difference that our agents do not only implement application-level coordination (synchronization, collection of intermediate results), but also management-level activities (local collection and filtering of load-balancing data), following the general approach we describe in [20]. As described in [19], control data generated by distributed resource control systems may be huge - and even higher in G-commerce systems, because of bidding and auctioning messages - and mobile agents may thus profitably be dispatched at the worker nodes for filtering the data flows at their source. We propose a further level of filtering to be accomplished by the deployment agents; this is even more necessary as we intend to control all three resources (CPU, memory and network). CPU is widely regarded as the most important factor. In [14] the authors propose to place worker agents within a Grid according not only to CPU load, but also to network bandwidth requirements; they relate a speed improvement of up to 40%, but the measurements were made in local-area clusters instead of dynamic sets of Internet hosts. Finally, memory control is usually ignored, but we contend that it has to be implemented in order to support typical scientific Grid computations, since they often imply storing and processing huge amounts of data.

Among the approaches that are not agent-based, the Globus initiative provides a complete toolkit addressing, among others, issues such as security, information discovery, resource management and portability. The Globus toolkit is being adopted as a standard by most multi-organisational Grids [10, 11].

7 Conclusion

Our goal is to customize computations on open Internet Grids. To this end, we believe that a Grid environment should provide high-level primitives enabling the reuse and combination of existing programs and distributed collections of data, without forcing the client to dive into low-level programming details; the Unix scripting approach is our model, and this translates into our abstract *deployment descriptor* proposal. From the implementation point of view, this translates into a mobile *deployment agent*, which synthesizes and enhances the benefits of several previous approaches: the deployment agent optimizes its own placement on the Grid, and consequently it reduces the overall load by minimizing the communications needed for application-level as well as management-level coordination. There are of course still some open questions. The first pertains to the actual efficiency of the proposed model, which cannot be entirely determined before the complete implementation of the distributed control mechanisms. The second concerns human factors such as validating the economical model (will the donator be able to earn real money?), or enabling the donator to decide on the lawfulness or ethics of computations submitted to him. This paper however concentrates on strictly technological aspects, and claims that the comprehensive combination of a *pure Java* implementation and a secure mobile agent platform is a unique asset for the portability, security and efficiency required for the success of Internet-based Grid computing.

References

1. A. Acharya, M. Ranganathan, and J. Saltz. Sumatra: A language for Resource-Aware mobile programs. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet, Second International Workshop*, volume 1222 of *LNCS*. Springer, July 1996.
2. G. Back and W. Hsieh. Drawing the red line in Java. In *Seventh IEEE Workshop on Hot Topics in Operating Systems*, Rio Rico, AZ, USA, March 1999.
3. Walter Binder. J-SEAL2 – A secure high-performance mobile agent system. In *IAT'99 Workshop on Agents in Electronic Commerce*, Hong Kong, December 1999.
4. Walter Binder. Design and implementation of the J-SEAL2 mobile agent kernel. In *The 2001 Symposium on Applications and the Internet (SAINT-2001)*, San Diego, CA, USA, January 2001.
5. Walter Binder, Jarle Hulaas, Alex Villazón, and Rory Vidal. Portable resource control in Java: The J-SEAL2 approach. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA-2001)*, Tampa Bay, Florida, USA, October 2001.
6. Walter Binder and Volker Roth. Secure mobile agent systems using Java: Where are we heading? In *Seventeenth ACM Symposium on Applied Computing (SAC-2002)*, Madrid, Spain, March 2002.
7. Ciarán Bryce and Jan Vitek. The JavaSeal mobile agent kernel. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, Palm Springs, CA, USA, October 1999.
8. P. Cerello and al. Grid Activities in Alice. In *International Conference on Computing in High Energy Physics 2001 (CHEP'01)*, 2001.

9. P. Evripidou, C. Panayiotou, G. Samaras, and E. Pitoura. The pacman metacomputer: Parallel computing with java mobile agents. *Future Generation Computer Systems Journal, Special Issue on Java in High Performance Computing*, 18(2):265–280, October 2001.
10. I. Foster and C. Kesselman. Computational Grids. In *The Grid: Blueprint for a Future Computing Infrastructure*, chapter 2. Morgan Kaufmann, 1999.
11. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid - Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.
12. J. Gomoluch and M. Schroeder. Information agents on the move: A survey on load-balancing with mobile agents. *Software Focus*, 2(2), 2001.
13. James Gosling, Bill Joy, and Guy L. Steele. *The Java Language Specification*. The Java Series. Addison-Wesley, Reading, MA, USA, 1st edition, 1996.
14. A. Keren and A. Barak. Adaptive placement of parallel java agents in a scalable computer cluster. In *Workshop on Java for High-Performance Network Computing*, Stanford University, Palo Alto, CA, USA, February 1998. ACM Press.
15. Andreas Krall and Philipp Tomsich. Java for large-scale scientific computations? In *Third International Conference on Large-Scale Scientific Computations (SCICOM-2001)*, Sozopol, Bulgaria, June 2001.
16. Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Reading, MA, USA, second edition, 1999.
17. N. Minar. Distributed systems topologies: Part 1 and part2. http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html, 2002.
18. J. E. Moreira, S. P. Midkoff, M. Gupta, P. V. Artigas, M. Snir, and R. D. Lawrence. Java programming for high-performance numerical computing. *IBM Systems Journal*, 39(1):21–56, 2000.
19. O. Tomarchio, L. Vita, and A. Puliafito. Active monitoring in grid environments using mobile agent technology. In *2nd Workshop on Active Middleware Services (AMS'00) in HPDC-9*, August 2000.
20. A. Villazón and J. Hulaas. Active network service management based on meta-level architectures. In W. Cazzola, R. J. Stroud, and F. Tisato, editors, *Reflection and Software Engineering*, volume 1826 of *LNCS*. Springer Verlag, Heidelberg, Germany, June 2000.
21. Alex Villazón and Walter Binder. Portable resource reification in Java-based mobile agent systems. In *Fifth IEEE International Conference on Mobile Agents (MA-2001)*, Atlanta, Georgia, USA, December 2001.
22. Jan Vitek and Giuseppe Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, 1999.
23. R. Wolski, S. Plank, T. Bryan, and J. Brevik. Analyzing Market-based Resource Allocation Strategies for the Computational Grid. *International Journal of High Performance Computing Applications*, 15(3), 2001.
24. F. Yellin. Low level security in Java. In *Fourth International Conference on the World-Wide Web*, MIT, Boston, USA, December 1995.