

# Communication Models for Processes and Services in Mobile Lightweight Grid Systems

L. Kirchev<sup>1</sup>, S. Isaiadis<sup>2</sup>, V. Georgiev<sup>1</sup>, V. Getov<sup>2</sup>

<sup>1</sup> Institute for Parallel Processing, Bulgarian Academy of Sciences  
{vasko, lkirchev}@acad.bg

<sup>2</sup> Harrow School of Computer Science, University of Westminster  
{s.isaiadis, v.s.getov}@westminster.ac.uk

## Abstract

*The last few years we have seen the emergence of pervasive and mobile computing technologies. At the same time the Grid has become the de facto standard for distributed and high performance computing. A lightweight Grid infrastructure may well provide a solid foundation for pervasive computing. For this to happen, we must rethink our design goals, and provide support for mobility, resource limited and non-dedicated environments, and redesign the process communication models to reflect the dynamic conditions of the system. In this paper we try to identify the requirements for supporting service and session mobility in lightweight Grid systems; review current approaches and discuss their limitations; and present our ideas for an integrated platform that meets these requirements.*

## 1. Introduction

Applications deployed on computational grids very often require communication facilities, as is the case, for example, with synchronous or interactive parallel applications. This necessitates the grid framework to offer relevant supporting communication mechanisms. An issue that is related to communication is the application task migration. In SOA environments, an application may be composed by a number of services in a workflow, so actually a task migration may be service migration as well.

Task migration may be triggered by a resource or service failure in the Grid infrastructure, or imposed by the load balancing mechanisms in the system.

The need for migration schemes becomes even more obvious in such dynamic environments as Grid systems comprised mainly of mobile devices. The latter are much more volatile than traditional Grid resources, as they are susceptible to a wider range of failures such as increased energy sensitivity due to battery issues; intermittent connectivity and unpredicted connection patterns due to the unreliable wireless links; and possible resource constraints due to size and mobility restrictions.

In this paper we take a fresh look at mobile and lightweight Grid infrastructures. Our perspective puts the mobile devices in the center and we discuss a set of requirements necessary to realize our conceptual architecture. We present a number of relevant projects in the field of checkpointing and migration and then propose our solution. Finally, we conclude this paper and lay the roadmap for our future work.

## 2. Relevant Work

**ProActive** [1] is a Java library for developing distributed and grid applications. It is based on the concept of the active object – i.e. object that has its own thread of execution. The application is built up of subsystems, any of which have a single active object and may have also passive objects. Outside the subsystem, only the active object is visible.

ProActive supports active objects' mobility, grouping of objects and establishing communications between the objects in the group, and also a SPMD programming model. Any active object in ProActive may migrate to another location (another JVM). The migration may be triggered either by the object itself, or by another agent. If the active object that migrates references passive objects, they will migrate with it too. After migration of an active object, messages sent to it are forwarded to the new location of the object.

ProActive offers a flexible mechanism for creating groups of objects and performing operations on the group as a whole. The results of such operations are also groups. The group communication enables simulation of an MPI-style collective communication. This can be used for implementing SPMD activities. Here an SPMD group is a collection of objects which reference all the other active objects in the group.

**Ibis** [2] project is a Java-based programming environment for grid computing. Central for its architecture is the Ibis Portability Layer (IPL), which offers an interface to different grid services such as monitoring, resource management, etc. It can have different implementations which may be selected and loaded at run time.

IPL provides the interface to various communication mechanisms. It provides one basic communication abstraction – unidirectional message channels. The endpoints of communication are send and receive ports, which are connected through these channels. For group communication, a send port may be connected to multiple receive ports, and vice versa. The actual communication may be realized through different protocols, such as TCP, UDP, MPI and specialized protocols such as Panda and GM, where supported by the interconnection. Ibis implements several application programming models on top of the IPL – RMI, whose implementation has some optimisations over the traditional RMI, group method invocation (GMI), which extends RMI with group communication, divide-and-conquer parallelism in the Satin system, and RepMI for communication through replicated objects (RepMI is under construction).

Although Ibis offers variable communication facilities, it does not have enough support for service mobility in highly dynamic environment. The Satin system offers migration, fault-tolerance and malleability, but for a specific type of problems – the divide-and-conquer tasks, while for a grid with clusters of mobile devices a more general model is needed.

**Agents based Grid systems.** Many grid systems use some form of mobile agents. As described in [3], in a grid system composed of desktop computers a user might be represented by an agent, which the user receives upon registering in the grid system. The user submits jobs in the system through the agent, which finds machines for execution of the job. If more machines are needed, the agent spawns child agents and sends them to different computers, where they start and monitor the user process. If a machine becomes unavailable, the monitoring mobile agent migrates the process on another available machine. For this purpose checkpointing is used.

For the purpose of process interaction, each process involved in the same job is addressed with a sequential number, valid only inside this job. Upon migration, this ID is translated into appropriate IP address. A Java wrapper object is in charge of supporting the process interaction. The user process calls the wrapper's message-forwarding method, passing the ID of the destination process. The message is passed to the corresponding mobile agent that tracks the machines, participating in the job execution. It maintains a table which translates ID to IP address. Upon migration the mobile agent sends the new address only to the processes that have communicated with the migrated process within a certain time interval.

**JGrid** is a Java and Jini-based grid system, developed at the Veszprem University, Hungary. The system supports process interaction based upon MPI-like message passing and high-level remote method calls [4], [5].

For the first one Java MPI method calls are used for communication. The Compute Services create the processes that perform the computations and establish socket connections between them. For the mapping between the physical channels and the logical connections a translation table is used. The mapping is transparent to the working processes. After starting the program, MPI\_init method initializes the communication infrastructure, thus connecting the processes through logical channels. In this way a wide-area parallel system may be created.

The second mechanism for process interaction allows tasks to communicate through remote method calls made through task proxies. When a remote process is created, the client receives a task control proxy, which references the task and may be passed to other tasks. Thus a set of remote tasks may store references to each other. Tasks can call remote methods on other tasks to implement a desired communication model.

Service migration is supported by the JGrid system through application level checkpointing – the service that requires migration support implements an interface with methods for saving the current state of the service. In case of migration the system is responsible for restarting the service on another host from the last checkpoint.

**H2O** [6] is a component-based and service-oriented framework, intended to provide lightweight and distributed resource sharing. It is based upon the idea of representing the resources as software components, which offer services through remote interfaces. Resource providers supply a runtime environment in the form of component containers (kernels). These containers are executed by the owners of resources and service components (pluglets) may be deployed in them not only by the owners of the containers, but also by third parties, provided that they possess the proper authorization. Inter-component communications are based on an extension to the Java RMI, called RMIX. It supports asynchronous calls, one-way calls and dynamic choice of the wire protocol.

H2O allows for the creation of a communication infrastructure that facilitates the execution of MPI programs across multidomain clusters. “Proxy” pluglets are used for transparently intercepting and forwarding the messages to the remote site. Proxy pluglets benefit from the H2O communication facilities for forwarding messages. They are loaded on all participating kernels and handles to these pluglets are distributed among all kernels. The communication between proxy pluglets is performed through direct channels between pluglets. The implementation of this infrastructure uses the MPICH library and makes some enhancements to it.

H2O does not offer direct support for service migration.

### **3. Requirements for Service Mobility**

The very different nature of mobile devices -and the subsequent integrated mobile Grid systems, mandates that we must rethink our design goals of the fundamental infrastructure layer of the Grid system. Supporting mechanisms in such environments are very different than traditional ones. We can identify mainly three classes of requirements in this case: monitoring related, group related and migration related requirements. In the following paragraphs we discuss in more detail these requirements.

#### **3.1. Monitoring**

As they say, prevention is better than cure, and so we must take actions to at least try to anticipate possible failures in our systems –if not prevent them altogether. Prevention is arguably a very challenging task, but in our case, diminishing the impact that a failure can have on our system is almost equivalent. For this to be realized, we need a monitoring scheme that will generate notifications in case a condition that might lead to a failure occurs in a resource or a particular service.

There are two classes of events that we need to monitor in such a system: hardware resource loads and connectivity issues. Hardware resources need to be monitored because they may lead to an overloaded resource and hence reduced performance. In an networked environment, reduced performance becomes the equivalent of a failed resource. Keeping tags on the loads of the most important resources –CPU, memory, network, battery, will allow us to take preemptive action.

Connectivity issues will eventually arise due to the mobile nature of the participating hosts. Trying to predict the mobility patterns of the resources/services might give us an opportunity to save and restore a task in a different node instead of helplessly waiting for the disconnection.

#### **3.2. Checkpointing / Migration**

Checkpointing and migration mechanisms become a necessity in very dynamic and unreliable environments. A mobile Grid system should support both transparent migration of services and application tasks in other available hosts. Neither the clients nor the service itself should have knowledge of the migration procedures.

In order for migration to be usable, checkpointing functionality should be supported as well. A snapshot of the executing task should be taken at periodic times, detailed enough for the migrated service to be able to continue from that point instead of starting all over again.

#### **3.3. Group Formation and Communication**

Formation of groups is needed for collective communications inside the mobile Grid system. Such communication is needed to identify and query available nodes; to collect information regarding the status of the nodes; to identify and select possible migration targets; and generally to realize and support the mobile service communication model. The formation of the groups should be truly dynamic –participating services should be able to join and leave the group at any time and for any reason.

#### **4. A Lightweight Grid Communication Model**

The key approach we see as crucial for meeting the aforementioned requirements is the grouping and clusterification of the non-dedicated and often inadequate resources in order to split the system functionalities in related layers and abstract away from prospective clients the mobile and dynamic nature of these resources.

In a previous work [7] we developed a communication model for a hierarchical lightweight grid platform. Its architecture consists of three layers. The different clusters which comprise the system represent the first level. The second level is the grid level which is comprised of inter-connected clusters. The (optional) third level is the ability to connect the platform with other grid systems. The entry point for both cluster and grid level is a portal. We have cluster portal for each cluster and a grid portal for the interconnected clusters. There are system services local for each cluster and they have corresponding representatives on the grid level.

The cluster level services are represented in the outer Grid through a number of aggregator services deployed at the cluster level proxy. One aggregator service is automatically generated and deployed for each group of similar service in the cluster (where similarity is determined by the published interface) These present a uniform and consistent interface to a dynamic set of underlying services. It also provides location and binding transparency which in turn gives us the flexibility to allow for many different binding protocols to be used for the cluster services e.g. Web or Grid services, RMI Activatable Objects and so on. The proxy knows all the details about the underlying nodes and hence clients wishing to communicate with services available in the cluster have to pass through the proxy. They will have no a priori knowledge of the topology, communication protocol, or binding address for the services –the proxy handles this procedure.

The communication among running services/tasks requires that they be uniquely identified. One obvious solution is to introduce unique ProcessID (TaskID) within the cluster and a ClusterID for each cluster in the grid. Thus, ClusterID+ProcessID points to a single running task. When Resource Management Service (RMS) receives a task for execution and allocates it to a particular node, it creates the mapping ProcessID-NodeID-ClusterID.

##### **4.1. Communication Management**

The platform uses centralized management (cluster-wide but not grid-wide) based on a system Communication Service (CS). CS performs similar activities to those performed by a communication meta-service with master-workers model. In this scenario messages are sent to CS for particular process and CS forwards them to the exact location. CS interacts with the Monitoring Service (MS) and RMS in order to update information about processes exact locations when they migrate.

The Node Service of each node intercepts the outgoing traffic from the node, and redirects it to the CS. (Here it is possible to present a wrapper service for the application service, which should intercept the outgoing messages, but this will complicate the infrastructure, that is why a better solution is this to be a function of the Node Service). After the NS intercepts a message, it adds to it the physical address of the node and forwards it to the CS. The CS delivers it to the destination. When the destination service

replies, its NS intercepts the reply and attaches to it its physical address. Thus any further communication does not pass through the CS, but takes place directly between the two services (actually, between their NS). For each message there is a time-out period. The Node Services of the communicating processes keep tables with the corresponding logical and physical address. They use these tables when the traffic goes directly and does not pass through the CS. If there is no reply after the end of the time-out, the corresponding record from the table will be removed. This may happen if the correspondent does not want currently to communicate, if it crashed or finished, or if it migrated. In this case the outgoing traffic should again pass through the CS, because if the process migrated, the CS would know its new location.

In this communication model, services are single-session in the sense that when multiple requests come for a single service multiple instances of the service are started, eventually on different nodes. Thus mobility of the session is automatically provided with the migration of the service – the checkpoint which saves the status of the service actually saves the session too. The checkpoint should also include the user token so that after the service finishes work the result is sent to the right user.

Group formation is supported by the aggregator services that represent a particular functionality [9]. Services in the cluster, once registered, they can join and leave the group on demand, and the indexing components of the platform always remain updated. Group communications are built on top of the mirrored and parallel interfaces that each aggregator service publishes. This collective layer on top of the resources “fabric” layer, allows the traditional semantics of mirrored invocation for higher reliability; and SPMD (Single Program, Multiple Data) for distribution of the load to many nodes in the cluster. The mirrored interfaces can be easily extended to support MPI-like collective operations like Gather, Scatter and synchronization barriers while Broadcast and filtered selective Multicast is already supported.

#### 4.2. Monitoring and Mobility Management

In order to monitor mobility, we are calculating the probability that a device will go off range in the next discreet time slot. We are using the straightforward and realistic assumption that the closer the device is to the access point, the better the chances it will remain in range [8]. Distance in this case is measured in milliseconds as the end-to-end communication delay between the node and the proxy behind the access point. This is in essence a “virtual” distance as it doesn’t depend on the actual distance but on the environmental conditions and obstacles between the two endpoints. In order to measure this distance we can use simple ICMP packets –i.e. ping utility. Then, let  $S_f$  denote the network’s standard range, the probability that a device  $n$  stays in range can be calculated as:

$$P_n^r = \frac{|S_f - S_n|}{S_f}$$

where  $S_n$  is the “virtual” distance of the device from the proxy.  $S_n$  can further be enhanced by taking into consideration previous measurements so that we try to “predict” the next distance. Assuming we keep track of the last  $j$  distances,  $S_n$  can be calculated as follows:

$$S_n = \sum_{k=1}^j W_k S_n^{j-k} \quad \text{where} \quad W_k = \frac{k}{C} \quad \text{and} \quad C = \sum_{k=1}^j k$$

where  $W_k$  is a weighting factor to give more importance to recent measurements.

We are making use of these calculations in order to prioritize the services in the local cluster indexing components. That is, the higher the probability a device will remain in range, the higher the chances it will be selected for an execution before other devices with lower probabilities. Furthermore, when detecting a handoff situation,

Of course the virtual distance is not the only measurement taken into consideration when monitoring the mobile devices: hardware resources like CPU, memory, network and battery should be monitored as well in order to anticipate possible failures due to lack of resources. For this, the intelligents installed in the devices, periodically provide information on the dynamic state of the host: CPU load, memory load, network load and battery levels. These periodic updates serve also heartbeats to determine “hard” failures – sudden hardware or communication failures that could not be predicted. After a certain timeout limit, the device is considered offline and action needs to be taken. Just like all features of the platform, heartbeats are totally configurable by the administrator. This ensures that different application needs are met in different installations

It is possible every service to have a personal intelligent, different from the one responsible for monitoring the status of the device. When the monitoring agent detects possibility of failure of the device, migration of all services on this node will be performed to another device in the cluster. On the other hand, if a failure of a single service is detected, which is not due to problems with the device, it will be restarted, either on the same or on another mobile node.

### 4.3. Checkpointing and Migration Management

Migration of a running task/process may happen in one of the following cases:

- ♦ the node, on which the task is allocated, fails and the task is moved to another node (for fault tolerance)
- ♦ the task itself fails
- ♦ there is a node with less load and the task is migrated to it (for load balancing)

Monitoring Service (MS) keeps track of running services and in case of service execution failure or node failure it informs RMS. RMS should decide what to do – restart the task from the last checkpoint or reallocate the task and restart it from the last check-point or the beginning. When a process is migrated, RMS updates the new location – the previous ProcessID is mapped to a new NodeID and a new ClusterID if the process is reallocated to another cluster. The latter may happen when the task was reallocated from Grid Resource Management Service (GRMS).

## 5. Integration within the STE Institute

The high-level system communication services supporting different types of communications in grids of non-dedicated or resource limited environments were not addressed in the roadmap of the STE Institute. However in this paper we addressed the problem of building a generic platform supporting system communications. We consider the clusterification and hierarchical proxies are the key approach for masking the various

types of service/task mobility caused by system requirements, application or resource failure.

## 6. Conclusion and Future Work

Our overview reveals that ProActive groups of objects is very close to our concept of building clusters that support the communication transparency between migrating services as well as on inter-cluster level. Another useful component of such integrated platform is also the Ibis IPL with its support of abstract communication channels. To achieve full mobility transparency these tools are to be upgraded with the system functionalities of our integrated platform such as mobility monitoring, checkpointing, migration and distant launching. A further step in our model is to consider the possibility of different levels of support of persistence – e.g. the communications of the persistent and non-persistent applications as well as communications for the multi-session or multi-instance services.

## References

- [1] Baude, F., Denis Caromel, Fabrice Huet and Julien Vayssiere. Communicating Mobile Active Objects in Java. *Proceedings of the 8th International Conference on High Performance Computing and Networking Europe*. May 8 - 10, 2000. Amsterdam, The Netherlands.
- [2] Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesinska, Rutger Hofman, Cerial Jacobs, Thilo Kielmann, and Henri E. Bal. Ibis: a flexible and efficient Java based grid programming environment. *Concurrency and Computation: Practice and Experience*, 17(7-8):1079-1107, June 2005.
- [3] Fukuda, M., Y. Tanaka, N. Suruki et. al. A Mobile-Agent-Based PC Grid, *Automatic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS '03)*, 2003, pp. 142-150.
- [4] Juhasz, Z., K. Kuntner, M. Magyarody, G. Major and S. Pota, JGrid Design Document, Department of Information Systems, University of Veszprem, available: [http://pds.irt.vien.hu/jgrid/doc/documentation/JGrid\\_Design.pdf](http://pds.irt.vien.hu/jgrid/doc/documentation/JGrid_Design.pdf)
- [5] Pota, S., G. Sipos, Z. Juhasz and P. Kacsuk, Parallel Program Execution Support in the JGrid System, *Distributed and Parallel Systems: Cluster and Grid Computing, Kluwer International Series in Engineering and Computer Science*, Vol. 777, Budapest, Hungery, 2004.
- [6] Dawid Kurzyniec, Tomasz Wrzosek, Dominik Drzewiecki, and Vaidy Sunderam. Towards self-organizing distributed computing frameworks: The H2O approach. *Parallel Processing Letters*, 13(2):273–290, 2003.
- [7] Lazar Kirchev, Minko Blyantov, Vasil Georgiev and Kiril Boyanov, A Communication Model Supporting Process Migration in Grid, *presented at EXPGRID, Paris, 21 June 2006*
- [8] S. Isaiadis, V. Getov, "Dependability in Hybrid Grid Systems: a Virtual Clusters Approach," in Proceedings of the IEEE JVA International Symposium on Modern Computing, 2006
- [9] S. Isaiadis and V. Getov, "A Lightweight Platform for Integration of Mobile Devices into Pervasive Grids", in Proceedings of High Performance Computing and Communications (HPCC), 2005.