Faculty of
Computer Science

universität
wien

Department of Distributed and
Multimedia Systems

Technical Report TR-20080403 April 2008

Towards a Service-Oriented Architecture (SOA) for Performance-Aware Mobile Grid Services

Karin Anna Hummel, Zoltan Juhasz

Towards a Service-Oriented Architecture (SOA) for Performance-Aware Mobile Grid Services: A Survey on Decentralized Performance Monitoring, SOAs, and Selected Meeting Application Use Cases for Mobile Devices

Technical Report: TR-20080403

Karin Anna Hummel
Dep. of Distributed and Multimedia Systems
University of Vienna, Austria
1010 Vienna, Lenaugasse 2/8
karin.hummel@univie.ac.at

Zoltan Juhasz
Dep. of Information Systems
University of Pannonia, Hungary
8200 Veszprem, Egyetem u. 10.
juhasz@irt.vein.hu

Abstract

When using grid technologies for scientific and non-scientific applications, ubiquitous access to these applications is useful for many scenarios. Hereby mobile devices provide a technology supporting this aim. By addressing multimedia integration into these services (like video streaming), the Quality of Service (QoS) is a major cause for user experience (QoE) and acceptance. The aim of this paper is to present a Service-Oriented Architecture (SOA) based on recent decentralized performance monitoring approaches and SOA based approaches able to support mobile grid clients. We first present a survey on decentralized performance monitoring and SOA and propose a general architecture. Additionally, we describe two application use cases including mobile devices which have been selected due to their different QoS constraints which require service adaptations due to dynamic changes in performance. These two use cases dedicated to the meeting application domain are: smart picture sharing and seamless video streaming.[1]

[1]This work has been funded by the WTZ Program Hungary/Austria 2006/2007. The technical report is published on http://www.cs.univie.ac.at/.

1 Introduction

By including multimedia services into Grids, the need for QoS assurance arises. To support these services, we present a performance-aware Grid architecture based on the SOA paradigm. We extend JGrid[7], a Jini based SOA, by introducing distributed performance monitoring and persistent storage of performance data in a shared communication space. As it is already included in the Jini technology, we propose the use of JavaSpaces as an example space. Hereby the approach for distributed performance monitoring can be centralized or decentralized. The latter avoids a single point of failure and thus increases robustness while having to solve the problem of gathering a common system-wide status.

The survey presented in this paper provides a summary of the two approaches relevant for our architecture: (i) decentralized performance monitoring and (ii) SOAs. Together with a summary of standard performance metrics applicable, this survey presents an introduction into performance-aware SOA Grid infrastructures.

We further provide the description of two example use case for SOA-based services which need QoS assurance and adaptation depending to the Grid's and the mobile client's performance status. The use cases are targeted to the business and meeting domain, they are:

smart picture sharing and seamless video streaming.

The remainder of this document is structured as follows: In Section 2 a summary of traditional performance metrics is given and their application to mobile Grids. In Section 3 we survey decentralized performance approaches and SOAs which leads to the concluding performance-aware SOA mobile Grid architecture presented in Section 4. Then we present a detailed description of the use cases benefiting from the approach in Section 5 and conclude our work.

2 Performance Metrics

According to the work of Jain [4], pp. 30 – 43, the two steps of performance evaluation are the selection of the evaluation technique and the metrics to use. The metrics are derived by distinguishing between three system states: (i) correct systems (providing correct answers and events), (ii) incorrect systems (providing wrong answers or events), and (iii) not responding systems. The corresponding classes of metrics are related to *Speed*, *Reliability*, and *Availability*. Figure 1 shows the hierarchical classification of the metrics used. In addition to the metrics commonly used for availability, we will further use the rates *number of connecting nodes* and *number of disconnecting nodes* in order to give a self-descriptive overview of the dynamics in a mobile grid system, that is a grid consisting of mobile nodes. In a system where the number of participating nodes is known and the failure is *disconnection*, these rates can be estimated by using the MTTF (Mean Time To Failure) and the MTTR (Mean Time To Repair). From a mobile distributed system's perspective, this rate is a major characterization of the dynamics of the system and thus, we will include them on the system level (see also the *churn rates* describing connection/disconnection rates in peer-to-peer systems).

By using these well-known metrics, we aim to derive high quality metrics in terms of low variability, non-redundancy, and completeness. Since we are investigating distributed systems, individual node metrics will be considered as well as global metrics.

In order to measure the speed of a (mobile) grid system, metrics related to time, rate, and utilization are used. Related to *time*, the following metrics definitions are used:

Response Time. Here, the *Response Time* is defined as the duration between the completion of a request by the user to the completion of the response of the system.[2] In case of a batch stream, the term *Turnaround Time* is used alternatively.

Reaction Time. The *Reaction Time* of a system is defined as the duration between the completion of the user request and the point in time the systems starts to execute the request.

Stretch Factor. This factor is used in order to describe the relation of the response time under a specific workload to the response time under minimal load.

The commonly used metrics describing *rates* are as follows:

Throughput. The *Throughput* is measured as *requests per unit of time* and varies depending on the load. Typically, the throughput increases with increasing load up to a certain point. Then the increase of the throughput might significantly slow down until it finally decreases or it decreases immediately. Depending on the system (and the system level of investigation), throughput is, e.g., measured as requests per unit of time (interactive systems), jobs per unit of time (batch systems), Millions of Instructions Per Second (MIPS) or Millions of FLoating Point OPerations (MFLOPS) (CPU), packets or bits per second (pps/bps) or Transactions Per Second (TPS) (transaction processing).

Nominal Capacity. Under ideal load conditions, this rate defines the maximum achievable throughput. For computer networks, the term *Bandwidth* is often used. In case the response time is too high when the nominal capacity is reached, the rate *Usable Capacity* refers to the maximum throughput with restricted response time.

Efficiency. The efficiency is defined as

$$Efficiency = \frac{Usable\ Capacity}{Nominal\ Capacity}.$$

[2] It is also possible to define the *Response Time* as the duration between the completion of the user request to the point in time, the systems starts to respond [4].

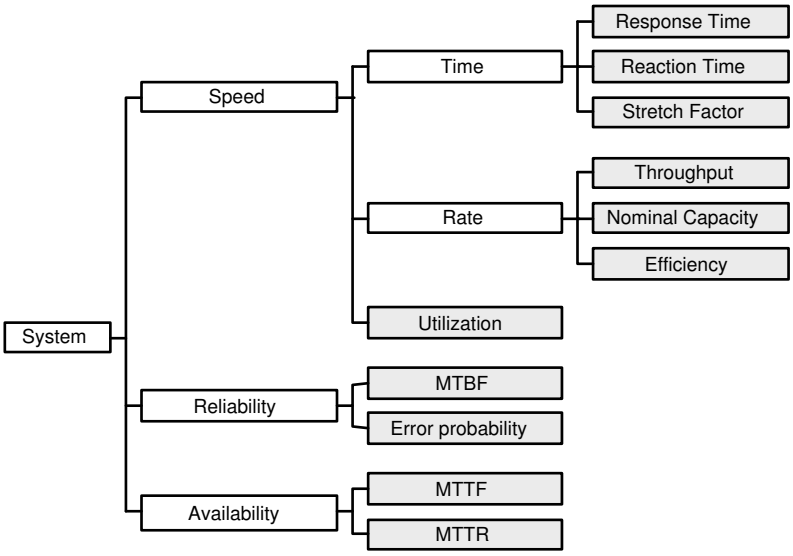

Figure 1: Performance metrics after [4].

Additionally, the utilization of a resource describes the workload processed during operation. This metric is defined as:

$$Utilization = \frac{Time_{busy}}{Time_{elapsed}}.$$

The reliability of a system describes the degree of confidence about correct operation of a system. Correct operation can be related to the value domain and the time domain. In particular for hard real-time systems, a late calculation may end in disaster as well as a wrong calculation. The *fault-hypothesis* describes the failures that are considered in the system. Independent of the type and semantic of a failure, the following two metrics can be used to describe a system's reliability:

Mean Time Between Failure (MTBF). This metric represents a time value which describes the average time between the occurrence of a failure.

Error Probability. This metric describes the probability of an error, that is, an observed system failure.

The availability of a system should be observed in cases where system may partially be unable to answer. In contrast to stationary distributed system where the availability is a measure of dependability of a system, in mobile systems, mobile devices are expected to roam in and out of the coverage area of wireless networks and, thus, to become unavailable from time to time. The availability metric values of a stationary grid system using wired lines are expected to differ significantly from their mobile counterparts. The following two metrics are considered:

Mean Time To Failure (MTTF). This time value describes the average time observed between the start of the observation period or a failure to the next failure.

Mean Time To Repair (MTTR). The mean time to repair describes the time needed for the system to become operational and responding again after a system failure.

Consequently, the availability of a system is described by the following ratio:

$$Availability = \frac{MTTF}{MTTF + MTTR}.$$

3 Related Work

Performance monitoring in distributed systems is an important issue as stated in [6] by presenting a market survey of real network management user's requirements. As a result, among the common appreciated

functions are the ability to monitor the complete system on a single console and the ability to use history information.

From an adaptive distributed Service-Oriented Architecture (SOA) point of view, these functions can be transformed into the desirable system requirements that SOA wide current and history performance data should be available to trigger the adaptation. If the system is decentralized, this aim might not be reached for 100 percent, but only approximated. In this section, related work in the area of network and service monitoring will be discussed (including decentralized approaches) as well as the second technology related to our work: SOA.

3.1 Distributed Performance Monitoring

In [6], most important *timing metrics* and *quantitative metrics* (statistics) are defined for system wide performance monitoring. Timing metrics are measured response times in total (from the time the client issues an request to the time the result is received by the client) and response times within the system itself like a middleware response time describing the time spent to process the client request in the middleware. Quantitative metrics are defined as the middleware throughput, the number of clients and servers active, (average) number of different messages exchanged, most used server (hotspot), and a deadlock count. An approach termed *Clearing House* supports these metrics for a data-base centric system consisting of exchange modules, which might be distributed. The architecture makes use of a central register facility for managing these exchange modules.

For scientific networking experiments, PlanetLab[3] is a well known testbed which provides networked computers on a world wide scale. Applications are run in virtual machines and share *slices* which can be reserved on selectable machines in the network. The administrators use slices as well for monitoring the system's performance on nodes in terms of workload (relation between *active* – i.e., containing a process – and *live* slices – i.e., in the past five minutes the slice has used at least 1% (300ms) of the CPU), CPU, memory, bandwidth, disk space, and jitter caused by scheduling latencies. To allow access for many users, PlanetLab implements a graceful degradation of the node's performance if a resource is becoming over-utilized.

For grid environments, in [5] a tool for testing and monitoring network performance has been introduced and developed as a part of the Globus toolkit[4]. This tool named *Gloperf* probes the network connecting grid nodes and calculates the bandwidth estimate accordingly (based on TCP end-to-end measurements). In order to keep the overhead for probing low, the tool supports a group membership which allows, for example, that a remote cluster is tested only once per group and not by all group member nodes. The overhead is further limited by allowing only a fraction of the available bandwidth to be used for probing. If this fraction is not sufficient for the probing frequency, the frequency is decreased. The group membership in this approach has been simply based on manual configuration.

The ASKALON framework [2] has been introduced to monitor the performance of scientific workflows by instrumentation of distributed grid management modules (engines). Workflow events can be monitored and reported to a central AKALON performance monitoring client. Example events are the initialization, suspension, and cancellation of activities.

When addressing decentralized systems, message overhead for the exchange of performance data should be minimized. In [3] the trade-off between communication costs and global performance, i.e., faults, in terms of false alarms and miss rates are analyzed in sensor networks. The approach of *virtual coordinate systems* allows to predict the latency in large scale networked systems like the Internet. Virtual coordinate systems allow nodes to map themselves into a coordinate systems and to derive distances to other nodes. Actual distance estimates only exist to nodes in a *reference set*. Latencies are calculated based on the distances in the virtual coordinate system which is usually either landmark based or decentralized [10]. In decentralized coordinate systems, the system can be described in terms of the reference set (e.g., neighbors in the network) the node uses to calculate the latency, the distance prediction mechanisms based on the distance definition (e.g., Euclidean distance), and an error minimization technique. In [10] attacks in a decen-

[3] see http://www.planet-lab.org/

[4] see http://www.globus.org/

tralized virtual coordinate system are investigated and mechanisms are proposed to increase the system's robustness.

To provide mechanisms for effective self-healing in distributed systems, performance problems have to be first localized. An approach based on Bayesian networks is proposed in [11]. Hereby, the inference model and simulation results are described without considering the problem of decentralization (and, thus, partial knowledge). Based on collected response time and elapsed time data, the system automatically infers elapsed times if current data is missing and estimates response times. Thus, it is possible to estimate the service causing the longest delays and apply self-healing mechanisms to these services.

3.2 Service Oriented Architecture

Service-orientation is a new paradigm in distributed software construction, rapidly gaining popularity in business and science applications. It is based on the notion of services (entities exporting some functionality through well-defined interfaces) and clients using these services for carrying out tasks. A crucial difference between client-server and service-oriented systems is the level of abstraction. In client-server systems, servers are physical resources represented by an IP address and port number. Services, on the other hand, hide implementation details behind their interface; the physical resource, its type and location typically is irrelevant to the user of the service.

A software architecture based on service-orientation is called a Service-Oriented Architecture (SOA). A service-oriented architecture can overcome several problems of classic distributed systems as we will discuss below. Figure 2 demonstrates a minimal service-oriented architecture. A service publishes its interface description in a directory. The information stored in the directory enables clients looking for a service with a particular functionality to connect and use the selected service. The use of the directory provides *naming* and *location transparency*, i.e., a service can be accessed without explicit knowledge of its address or location.

The use of service interfaces decouples service functionality definition from its implementation. Clients and services are loosely coupled in the sense that changes in the service implementation will not affect clients.

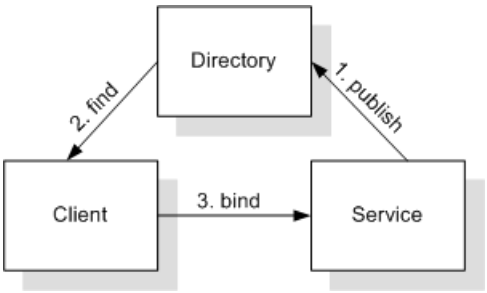

Figure 2: A minimal service-oriented architecture.

Service-orientation is only a concept. It requires a physical implementation technology to be usable. The two most well-known such technologies are Web Services, and Jini Technology. Web Services Technology [1] is a language-independent SOA environment due to its reliance on the XML language. A web service describes its interface and invocation details in an XML (Web Service Description Language) document stored in a UDDI directory[5]. The advantage of this scheme is that clients written in arbitrary programming languages can invoke services implemented in different languages. Clients can invoke services by sending SOAP[6] messages either by using Remote Procedure Call or message passing semantics.

Jini Technology [9] provides a service-oriented architecture for the Java Platform. Jini services are described by Java interfaces; clients use these interfaces to look for suitable services. Central to Jini is the notion of discovery. Participants in a Jini community use multicast or unicast messages to locate available Lookup Services (service directories) on the network. Lookup services are used to register and look up the services of interest. Jini relies on object mobility in its operation. Java objects are passed to the Lookup Service during registration and this object is forwarded to the client as a lookup result. The downloaded object is used as a service proxy in the client enabling connection and service invocation in - typically - Remote Method Invocation fashion.

Jini has been designed for dynamic communities in

[5] see http://www.oasis-open.org/specs/index.php#uddiv3.0.2
[6] see http://www.w3.org/TR/soap/

which services and clients join and leave the system in unpredictable ways. Resource management is a problematic issue in such environments. Jini uses the notion of leasing to address these problems. Resource usage (e.g., service registration in the Lookup Service) is normally allowed on a timed basis. Entities must renew their lease period before it expires if they need to continue with that activity. If the lease is not renewed – either deliberately or due to system failure – the corresponding resource is released (e.g., registration object deleted). This scheme is well suited to systems with unreliable network connection and intermittently operating devices.

4 Architecture

The architecture proposed should support the adaptation of services due to QoS achievable within the grid. In the following sections, performance-awareness is introduced to the SOA JGrid [7].

4.1 JGrid

JGrid [7] is an experimental service grid framework developed at the University of Pannonia in collaboration with MTA SZTAKI and Eotvos University Budapest. Its aim is to create a reliable environment in which various services provide capabilities to clients in a reliable way despite the geographical disparity, presence of failures, and the dynamically changing topology.

It is built on top of Jini Technology, hence relying on its discovery, leasing and platform independence features. There are a set of core services that provide fundamental functionalities for the system. The *Authentication Service* is responsible for identifying grid users and allowing access for them to the system. The *Registration Service* is used to store access right-user role pairs for services. This is an extension to the security architecture of Java. *Compute Service*s are special entities that enable clients to execute Java objects (sent from the client program) on a remote location. The Compute Service is highly configurable and supports a number of sequential and parallel execution modes. The *Batch Service* is used as a wrapper service to export native batch execution environments as services in order to provide traditional batch execution facilities in JGrid. Finally, the *Storage Service* provides access to remote files and directories through a service interface.

The core JGrid services along with the standard Jini services, such as Transaction, Event Mailbox, JavaSpaces, etc services form a rich platform for creating dynamic service-oriented applications. Several case studies demonstrate the success of this approach including video stream, Internet radio, weather instrument services, and computationally intensive applications using several compute services.

An important feature of JGrid with respect to performance monitoring and providing QoS is its ability to provide static and dynamic information about the service's capabilities and performance. Jini attribute objects are used to describe static properties, e.g., processor speed, memory size, network bandwidth, screen size, etc., that clients can use in service selection decisions. Services can be assigned so called monitor proxies that clients can retrieve via the main service proxy to monitor dynamically changing performance metrics, e.g., latency, processor load, etc.

4.2 Introducing Distributed Performance Gathering to JGrid

This section presents a performance framework which defines the metrics used to evaluate three different levels of a mobile SOA grid: the mobile client's perspective, the system's perspective, and the grid node level's perspective including connection characteristics. The main three questions arising are:

- ◇ What should be measured?
- ◇ Where and how should the measurement take place?
- ◇ How can the measurements be stored and distributed in a mobile and decentralized environment?

We propose a generic approach, thus, metrics can be easily added. However, we will start with a sample set of the following metrics necessary to trigger adaptations for multimedia services QoS: available network bandwidth, CPU utilization, availability in terms of MTTF and MTTR, and reliability in terms of the MTBF.

The approach proposed is based on the space-based paradigm (which can be supported by JavaSpaces included in Jini but also implemented on top of Jini and the communication infrastructure), the measurement results are stored by means of the space. Depending on the decomposition level, the following views can be defined:

Node view. Here, the metrics are applied to each single grid node. In order to describe network connection as point-to-point links, which are needed to describe ad-hoc networking mode, a matrix is used to describe the link performance measures as depicted by Table 1. This matrix represents also the topology of the node network.[7] Matrices are very useful to describe the links between nodes and, for example, have already been used in performance oriented grid research, like the *network weather service* presented in [8]. Usually, this matrix will be symmetric due to the symmetry of the link.[8]

System view. Here, the metrics are used to describe the grid as a whole. The measurements on node level are aggregated in order to provide the system view metrics for the *collective layer*. For example, the variance of node utilization might be a useful means to determine under-utilized resources. Additionally, the overhead caused by brokering and scheduling can be observed on system level.

Client view. From the client's perspective, the application's performance characteristics are important. These metrics are applied accordingly and represent the *application layer* point of view. Furthermore, selected metrics are used to describe the mobile grid client itself, like, for example, the disconnection frequency.

The measurements take place on different nodes and should be stored persistently. Here, both time series, that are, the *history of of measurements* and the *current status* are considered. We address performance evaluation completely on software layer by means of code instrumentation and generation of logs.

[7] Note, that this matrix might be sparse for ad-hoc networks.

[8] In case of, for example, satellite links, up- and down-link capacity differ.

	$Node_1$	$Node_2$	$Node_3$	...	$Node_n$
$Node_1$	x_{11}	x_{12}	x_{13}	x_{14}	x_{1n}
$Node_2$	x_{21}	x_{22}	x_{23}	x_{24}	x_{2n}
$Node_3$	x_{31}	x_{32}	x_{33}	x_{34}	x_{3n}
...	...	...	...	...	...
$Node_n$	x_{n1}	x_{n2}	x_{n3}	...	x_{nn}

Table 1: Connectivity performance matrix

Each JGrid node is responsible for generating logs on node level. The JGrid management which includes brokering and scheduling is responsible for measurement and generation of logs in terms of system metrics. The system measures will be executed in a centralized or decentralized manner depending on the system management policy. The client measures will be generated at the client side. However, these measures should be available to the monitoring framework for evaluation purposes.

We propose to use the persistent layer provided by the space-based paradigm for storing the measures. Both JGrid clients and JGrid nodes may disconnect from the grid while the measures which have been generated previously should remain available. Since JGrid is based on Jini, we propose to use JavaSpaces for persistent storage of performance data. Figure 3 depicts the principle of using the space for logging purpose. The log information should contain meta-information about the measurements, which can easily be achieved by using XML.

5 Use Cases

We will now define two use cases for mobile grids for SOA based applications exhibiting different QoS requirements, they are: smart picture sharing and seamless video streaming.

Smart picture sharing. In business and technical meetings, it is generally a problem to easily share presentations and pictures during the meeting. Copying slides to the computer driving the projector or switching projector cable from one laptop to the other are cumbersome (although, this should in principle work). We envision a service-oriented picture sharing environment that greatly reduces the above complexity and

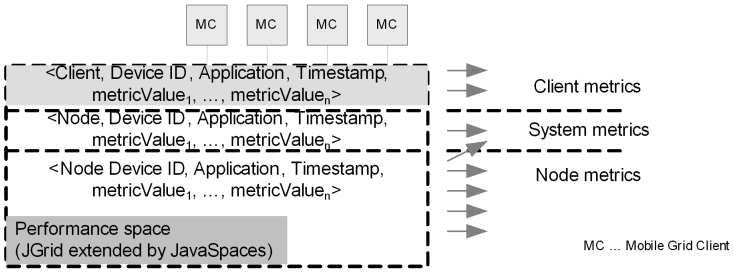

Figure 3: Logging of performance measures in JGrid extended by space-based technology.

enhances meetings with additional collaborative functionalities.

Participants can use PDA or laptop devices with client programs with which they can discover the projector service as they enter the meeting room. The projector service provides presentation and picture display facilities for each user. Access to the projector can be controlled manually or by a policy component that grants the display right to the requesting client base on pre-set rules.

If required, clients can request the currently displayed picture to be transferred from the projector service to the client device on which they can place annotations or add drawings. These can be displayed on the projector if so wished.

The devices and the projector exchange QoS parameters (most importantly screen resolution and color depth) in order to perform automatic conversion and transformation during picture transfer.

The service-oriented nature of the environment allows the use of other services; e.g., one can connect to his or her remote file service to load a presentation or picture not available locally on the device. Another possibility is to use a shared white-board service or a printer to print out pictures or notes.

Communication network parameters can be monitored during operation to change, e.g., picture quality if communication bandwidth is not adequate for the current image format. Similarly, image conversion beyond what is required for the devices screen size could be triggered if, e.g., the memory constraints of a PDA device must be met.

In this scenario, important characteristics of the service quality are:

- ◇ Latency caused by auto-configuration
- ◇ Response time for the service (when sharing, projecting, updating pictures, slowing down the slide show, etc.)
- ◇ Latency and jitter effects for the slide show.

A first prototypical implementation of the picture sharing scenario on PDAs shows the feasibility of accessing this JGrid service via mobile devices.

Seamless video streaming. The streaming of video should be seamlessly supported. The user should be able to watch a video on the device best fitting and change the output/input devices while watching. For example, when moving around, the video might be streamed to a PDA. When the user enters the working room, the video will be displayed on the PC screen. When the user enters his or her own private household's living room, the video should be displayed on a wall display. Here, the seamless migration of the video is demanding.

Important quality characteristics are:

- ◇ Video quality, that is, degree of absence of the following faults: artifacts, stalls, blurring, added edge energy, "mosquito noise"
- ◇ Service and network quality: latency and jitter effects, packet loss, and response time.

Figure 4 depicts the architecture of a smart business environment where the applications described should be embedded. Mobile devices of different types work together with services provided by smart devices like a smart bookshelf capable of tracking books that are put into or taken out of the bookshelf enhanced by RFID technology, and beamers. Mobile devices are expected

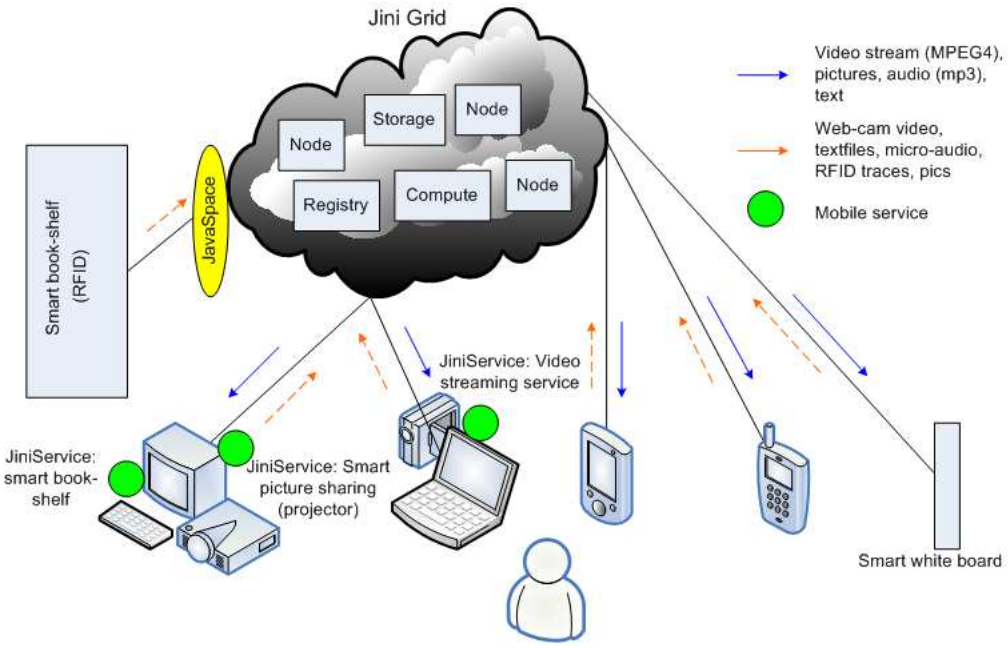

Figure 4: Service architecture.

to act as interfaces to Jini services and the JGrid infrastructure provides computing services for computing intensive tasks (like video coding). Hereby, the quality of the services provided should be permanently monitored to allow automatic system adaptation. Each component should be self-descriptive in terms of classic performance metrics (like utilization, response time, availability, etc.), and thus, the performance measurement framework emerges within the distributed system. The availability of the framework may be increased by replicating the performance information or services.

6 Conclusions

In this paper, we have surveyed decentralized performance monitoring approaches and described the main aspects of Service-Oriented Architectures (SOAs). We have further proposed a performance aware architecture which generates a system view based on aggregating node performance measurements. For persistent performance data storage we proposed to use space based approaches. We applied the general concept to JGrid, a Jini based SOA. We further defined and described two use cases for multimedia services: *smart picture sharing* and *seamless video streaming*.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag, 2004.

[2] P. Brunner, H.-L. Truong, and T. Fahringer. Performance Monitoring and Visualization of Grid Scientific Workflows in ASKALON. In *2nd Int. Conference on High Performance Computing and Communication*, 2006.

[3] E. Ermis and V. Saligrama. Adaptive Statistical Sampling Methods for Decentralized Estimation and Detection of Localized Phenomena. In *4th Int. Symposium on Information Processing in Sensor Networks*, 2005.

[4] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1992.

[5] C. Lee, R. Wolski, C. Kesselman, and I. Foster. A Network Performance Tool for Grid Environments. In *1999 IEEE/ACM Conference on Supercomputing*, 1999.

[6] J. McCann and K. Manning. Tool to Evaluate Performance in Distributed Heterogeneous Processing. In *6th Euromicro Conference on Parallel and Distributed Processing*, 1998.

[7] S. Pota and Z. Juhasz. High-Level Execution and Communication Support for Parallel Grid Applications in JGrid. In *20th IEEE International Parallel and Distributed Processing Symposium*, 2006.

[8] M. Swany and R. Wolski. Building Performance Topologies for Computational Grids. *International Journal of High Performance Computing Applications*, 18(2), 2003.

[9] J. Waldo and K. Arnold. *The Jini Specifications*. Addison-Wesley, 2000.

[10] D. Zage and C. Nita-Rotaru. On the Accuracy of Decentralized Virtual Coordinate Systems in Adversarial Networks. In *14th ACM Conference on Computer and Communications Security*, 2007.

[11] R. Zhang, S. Moyle, S. McKeever, and A. Bivens. Performance Problem Localization in Self-Healing, Service-Oriented Systems Using Bayesian Networks. In *14th ACM Conference on Computer and Communications Security*, 2007.