Resource Management with Stateful Support for Analytic Applications

Liana L. Fong[1], Catherine H. Crawford[2], Hidayatullah Shaikh[1]
[1]IBM T. J. Watson Research Center
[2]IBM System Technology Group
{llfong, catcraw, hshaikh}@us.ibm.com

Abstract

Analytic applications from various industrial sectors have specific attributes and requirements including relatively long processing time, parallelization, multiple interactive invocations, web services, and expected quality of service objectives. Current parallel resource management systems for batch-oriented jobs lack the effective support for multiple interactive invocations with consideration in quality of service objectives, while transaction processing systems do not support dynamic creation of parallel application instances. To better serve the analytic applications, a set of additional resource management services, defined as stateful support, introduces the concept of Service Instance and Service Instance Management. This set of stateful support services can be implemented as extension to existing parallel resource management to serve these analytic applications that rapidly increase in the demand of computing power.

1. Introduction

Analytic applications are broadly referred to as software to assist decision making or action navigating based on data possibly from various sources. The use of computing power for executing these analytic applications is growing in many different sectors such as government, medical, automotive, and etc. One of the well publicized computing infrastructures used for running financial analytic applications was JP Morgan Chase's Compute Backbone that was planned for combining up 1000 CPU in a grid system environment [19]. Our work will address some resource management issues related to analytic applications.

In this paper, we first describe a set of specific attributes and requirements existing in these classes of analytic applications from different industries. Based on these attributes and requirements, we justify the introduction of a generic set of resource management services, which we refer to as stateful support services, to better support many of the analytic applications. These services are designed as an extension to current resource management systems (also known as resource scheduling systems).

1.1. Motivation for Stateful Support

In various market segments, many customers desire better than current support in their analytic applications to gain competitive advantage in programming model, execution models and differentiating quality of services without large increase in the total cost of ownership on system infrastructure. These analytic applications have unique or combinations of attributes that are different from transactional and legacy batch workload. These attributes, described in details below, include relatively long processing time, parallelism, stateful invocation, and invocations with quality of service objectives.

Unlike transactional workload, analytic applications usually have relatively ***long processing time*** requirements. Instead of sub-second processing time, applications may run in tens or hundreds of seconds. Processing time of different applications likely has high variance. Moreover, the processing time of the same application may vary greatly with different input parameters. For example, the processing time for pricing American option of an asset portfolio with 5 entries took approximately 380 seconds while a portfolio of 7 entries took about 518 seconds with the same mesh size of 1024 and running on a single processor [2].

While analytic applications can benefit from high speed processing power and I/O as well as large memory sizes, some applications can also leverage ***parallelism*** to optimize execution across processing systems in clustered infrastructures. In term of parallelism, there are similarities of these applications to high performance scientific parallel batch workload. The term "embarrassing parallelism" is used to describe a common technique consisting of splitting up a large piece of work into a set of independent execution units to take advantage of multiple processing units in clustered environments. Many analytic applications can take

1-4244-0054-6/06/$20.00

advantage of "embarrassing parallelism". Both "push" and "pull" work assignment models between the scheduling management function and the work processing units are applicable. For parallel applications that require communication or data exchanges between parallel processing units, the use of MPI (message passing interface) prevails in scientific applications. The trend is less clear that the use of MPI would become increasingly common among analytic applications. Until there are new common parallel programming and communication models, the support for MPI will be necessary. Launching a parallel application, using MPI or not, has considerable overhead. There are costs in using a scheduling management function to allocate multiple resources, and costs in establishing connectivity to the parallel tasks.

Customers often express strong requirements in a low-latency infrastructure to support their analytic applications. These requirements go beyond the hardware infrastructure such as high speed network connections and high power servers in clustered environments. Minimizing the launching of applications costs would help lower the infrastructure latency. Many analytic applications often also require permission to use certain software licenses, establishment of many connections to data sources, and/or loading of large amounts of data and library routines into memory as working sets. In addition to launching costs, there is great advantage of maintaining the permission, connections, data, and routines for many invocations to the same application such that all these costs can be amortized across invocations. Information for permission, connections, data, routines, resources allocated, and others is generally referred to as infrastructure "states" of applications. A system management that can support multiple ***stateful invocations*** is therefore desirable for analytic applications to achieve low-latency and high performance execution.

Analytic applications can be scheduled to run on systems like other scientific high performance batch-oriented jobs. However, there is an increasing demand in ***interactive invocations,*** similar to transaction requests, of these analytic applications. Interactive invocations also provide users the ability to examine results and steer the direction of the subsequent invocations. Instead of batch-oriented job submissions, invocations of these ***applications as services*** would also be desired. By defining the appropriate service interfaces for these applications, one would reduce the complexity of application deployment and invocation, including dynamic resource allocation, and would also enhance the interoperability of these applications.

Unlike batch-oriented job execution, the demand for ***quality of services*** of analytic applications as interactive invocations would be similar to transactional execution with response time goals, or/and to real-time application execution with completion deadlines. The quality of services for interactive invocations affects greatly the productivity and satisfaction of users. However, achieving good quality of services should also be balanced with the total cost of ownership of the system infrastructure. Without adequate management on resource allocation and appropriate control of application service instances, achieving quality of service can lead to over-provisioning of services, causing increased total cost of ownership.

After describing the general attributes and requirements of analytic applications, we will explore some concrete applications that exhibit these attributes in the next section.

1.2. Numerical Simulations in Capital Markets

In the financial industry, workload, such as Value at Risk (VaR) and Profit and Loss reporting of portfolios, is both data and computation intensive. For these computations, typically a stochastic time integration technique such as Monte Carlo, can be parallelized by splitting up the scenarios for a portfolio and loading various data sets associated with deals onto the computation nodes in a clustered system environment. The parallel calculations can possibly yield results in few seconds for a portfolio instead of ten of minutes without parallelism. Explorations of parallel techniques to reduce response time for such calculations are many [2, 11]. As to the previous example on the processing time for pricing the American option of an asset portfolio with 5 entries, the job took about 380 seconds on a single processor, but only took about 16 seconds on a SGI Origin with 32 processors [2]. Also, when computation power posts no usage constraint, the application users have the options in generating more accurate results by increasing the iterations of calculations or even using better, yet more computation intensive, simulation models.

Moreover, the effort in reducing the response time of these calculations also reflects the demand of changing the role of risk management and portfolio analyses from the middle-office function, primary for end-of-day reporting and over-night batch processing, to the online, real-time valuation and user facing instruments used by market analysts or traders. To further push the technologies in improving response time from multiple seconds to sub-second, one can use methodologies of incremental calculation. However, increment calculations require storing the results of individual simulations or parameters necessary to reproduce the results at an invocation or portfolio/sub-portfolio level [16]. Applications, like incremental calculations,

motivate our design in building system support for multiple and stateful invocations of applications.

The long-running calculations and real-time valuations outlined here are often executed on the same cluster of resources. The resources must be shared and allocated across the cluster in a manner that is consistent with the overall business values that each calculation provides. Specifications of quality of service objectives and optimization of resource allocation to meet the quality of service objectives are topics of many studies for batch oriented systems or transactions. However, the need to optimally meet the quality of services for mixed interactive and batch parallel applications with stateful invocations is likely to impose additional constraints for the resource optimization problems.

In summary, we described that the analytic financial applications are computation intensive, and can be benefited from paralellization, stateful invocation support, and differentiating quality of services.

1.3. Image Analysis and Modeling in Medical Industry

Medical image analysis and modeling are essential to the medical industry [14]. More than ever, computer power is used to store and manipulate the tremendous amount of image data generated from medical devices (MRI, CT scanners). A single radiology department may produce tens of terabytes per year. The use of federated data within or across medical institutes assists medical researchers in statistical and epidemiological studies. Computer systems are also used to generate 3D medical images that are more realistic representation of our 3D human organs and functions than 2D imaging data produced from devices such as scanners. The use of 3D medical images generated from real medical data has the promise to assist clinicians in more timely diagnosis and appropriate treatment. Simulated medical images based on both hand and automated segmentation would be potentially useful to create virtual anatomy for education and training purposes [10].

Applications for medial images analysis and modeling exhibit many of the attributes and requirements that are discussed in the Motivation [Section 1.1] of this paper. The applications are often computation intensive (e.g. image registration, volume reconstruction). For example, a volume reconstruction step in the PTM3D image analyzer took about 20 minutes in processing time on a PC system of 3GHz speed and 2MB memory [8]. Parallelism can benefit these applications [1, 8, 15]. The processing time of the image analyzer example cited here was reduced to less than 2 minutes when run on a grid cluster. These applications also require access to large amount of data. For example, the image processing applications for the multiple sclerosis treatment trial [5] potentially accessed terabytes of data [15]. The cost in establishing permission to the data, creating connection and/or loading can be high. It would be desirable to amortize the cost over multiple iterations with users input. More importantly, the need to support interactive tasks is critical for these applications. The human interactions are required for reasons of legality, treatment decision, avoiding local minima in high-degree-of-freedom optimization problems [8, 9, 15]. Unlike batch oriented jobs, the responsiveness requirements of these interactive applications are similar to the requirements of transactional workload. Projects AGIR and GRaDS [6, 8] were initiated to build application level schedulers to meet the specific needs in the data management, interactivity and quality of services (e.g. soft real-time scheduling) of medical image applications. The use of web services and grid middleware is explored in these projects.

2. Resource Management Support for Stateful Invocation

There are many mature resource scheduling systems that support resource allocations for parallel batch jobs with sophisticated algorithms in optimizing the usage of resources in clustered environments [12, 13]. However, none of them supports our defined concept of stateful invocations of parallel application services. For each invocation of an application instance, these scheduling systems assume that resources are allocated in creating a new instance of the application and resources are released at the termination of the invocation.

There are also autonomic management systems that support dynamic deployment of multiple instances of application services (e.g. IBM WebSphere Extended Deployment [21]). The application service instance would stay as long as there are demands for such services. However, the multiplicity of these application instances is deployed to meet the demand and load balancing of requests, not as instantiations of a parallel application service. Some of these systems support affinity routing such that sequenced requests are routed to the same instance of an application. The ideal of affinity routing can be viewed as a weaker form of stateful invocation.

In the following sections, we present the design of a set of resource management features to support stateful invocations for the analytic applications.

2.1. Stateful Support Components

We envision that this set of management features is an enhancement, and not as a replacement, to the existing resource management systems that support both serial and parallel applications. In Figure 1, we introduce the

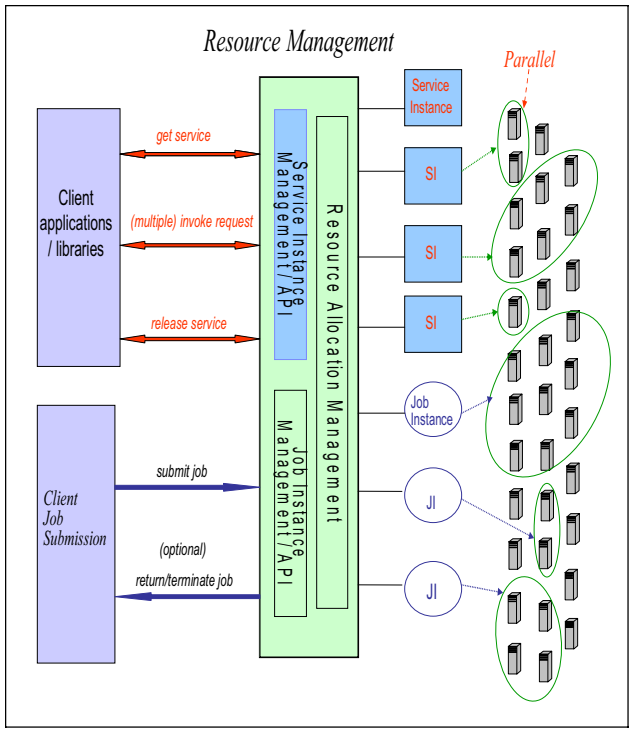

Figure 1: Stateful support enhancements to resource scheduling system

construct of Service Instance and the functional component of Service Instance Management (in color magenta) and their relations to an existing generic resource management system which has functional components such as job instance management and resource allocation management.

A Service Instance (SI) is an application service running on a particular set of resources, loaded with application specific objects and libraries. The Service Instance Management (SIM) component has the responsibility in managing the lifecycle of a Service Instance in the system. SIM interacts with the Resource Allocation Management component to acquire and return sets of resources at the creation and termination of SI's. The SIM also supports a set of service interfaces which clients can use to interact with our system. The next section details the service interfaces.

2.2. Service Abstractions

Our stateful support design provides two categories of service abstractions: application services and scheduler services.

2.2.1. Application Services

Our project assumes that analytic applications can be structured as application services. Thus, the "how-to" on structuring applications as services is outside the scope of this paper. Application services can be either stateless or stateful. Web Services is the technology of choice for stateless applications with loosely coupled clients and servers. A WSDL document can represent the interface of a stateless application services. For stateful web services, we use the Web Services Resource Framework (WSRF) specification [20] which is the emerging OASIS Web services standard for modeling and accessing stateful resources, using Web services. The interface of a stateful application service can also be represented by a WSDL document. However the WSDL document must include additional operations as specified by the WSRF specifications e.g. operations for accessing the state of the resource as resource properties.

Instances of these application services can be created using application specific scripts and/or programs onto one or a group of node resources. We also refer the initialization of the application services onto the resources as application provisioning. The management of these applications as service instances and the associated resources, such as servers, required to host the services are in the functional scope of the scheduler services (details in following sections).

2.2.2. Scheduler Services

The scheduler services manage the lifecycle (i.e. creation and termination) of stateful application services and also provide query interface to locate a particular instance of a stateful application service. The scheduler services for stateful invocations are a generic set for any scheduler, though the design implementation can be resource management system specific. These services create and use service handles to identify specific application service instances. The WSRF specifications refer to these stateful resource handles as Endpoint References (EPR).

Our current design of this generic set includes the following services:

- *getService* - given a service type, return an EPR to a service instance, which is obtained by allocating necessary resources
- *releaseService* – given a EPR, the designated service instance is released from current usage; resources allocated to the service instance will also be released
- *queryService* – query the service instances managed in the Service Instance Management and return EPRs based on matching criteria

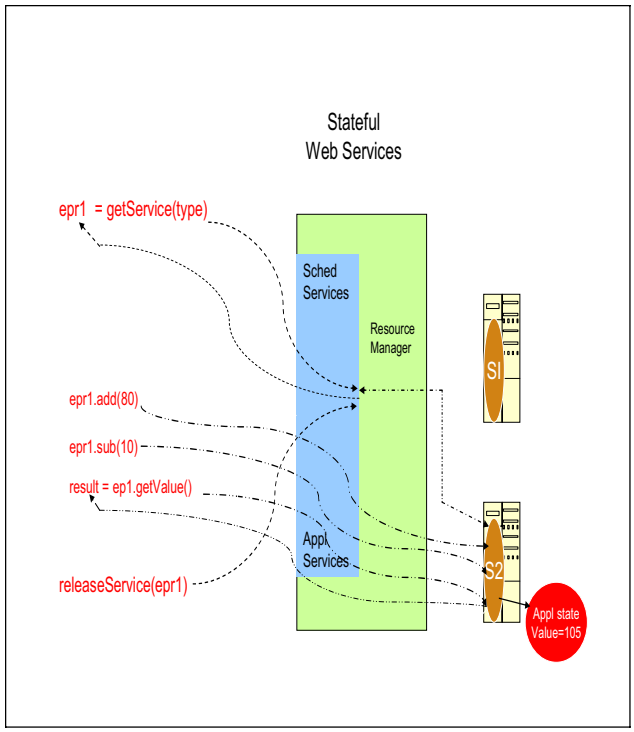

Figure 2: Exemplary stateful application and scheduling Services

Figure 2 shows a sample invocation of various operations on the scheduler and application services. The application service is a simple stateful Calculator service that exposes operations such as add(), sub(), and getValue(). The invocation of the *getService* scheduler service returns the EPR to an instance of Calculator service. A sample EPR for the Calculator service is shown in the listing below:

```
<wsa:EndpointReference>
   <wsa:Address>
      http://www.ibm.com/samples/CalculatorService
   </wsa:Address>
   <wsa:ReferenceParameters>
      <calc:ServiceInstanceId>
         ServiceInstance123
      </calc: ServiceInstanceId>
   </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

The EPR contains the URL of the web service identifying the service type and also include the identifier (ServiceInstanceId) that uniquely identifies the service instance. This EPR can then be used to perform stateful invocation of any operation on the application service. Any stateful invocations of operations on the service instance will carry the service instance identifier in the SOAP message. The system uses this identifier to route the service request to the correct service instance. A sample SOAP message for invocation of the "add" operation on the calculator service is shown below:

```
<soapenv:Envelope xmlns:S=
   "http://schemas.xmlsoap.org/soap
   /envelope/" xmlns:wsa="...">
<S:Header>
   <wsa:to>
      http http://www.ibm.com/samples
/CalculatorService
   </wsa:to>
   <calc:ServiceInstanceId
wsa:IsReferenceParameter="true">
      ServiceInstance123
   </calc: ServiceInstanceId>
</S:Header>
<S:Body>
      <add xmlns="http:
      //calculator.ibm.com">
      <value>100</value>
   </add>
</S:Body>
</soapenv:Envelope>
```

Once the Calculator service instance is no longer needed it can be terminated by using the *releaseService* .

2.3. Service Instance Management Design and Implementation Considerations

In the previous sections, we describe the application and scheduler services abstraction, and also provide a usage example of these services. In this section, we will discuss the detailed design and implementation of two scheduler services: *getService* and *releaseService*, which are services more relevant to the resource management than other services such as query.

To support *getService* and *releaseServcie*, the Service Instance Management can have designs and implementations of various levels of sophistication in functions along two dimensions. One dimension is along the sophistication of interactions between SIP and Resource Allocation management when allocating and deallocation of resources for service instances. The second dimension is on the provisioning and de-provisioning of the service instances with application specifics for the *getService* and *releaseService*. On the simplest level for both dimensions, SIM simply maintains a list of existing service instances and their usage status. The *getService* function will do a search for a not–in-use

services instance of the matched service type and will then mark it in-use before returning the caller the ERP of the service instance. The *releaseService* will change in-use status to not-in-use for a service instance. In this way, creating/deleting and provisioning/de-provisioning of service instances are outside the control of SIM. This simple design and implementation is useful for functional testing of the service interfaces.

In our research project, we designed our experimental SIM model with more functions. Figure 3 shows the interactions of SIM with Resource Allocation Management, to allocate resources, and Application Provisioner, to initialize service instance with application specifics, when *getService* is processed. Moreover, SIM also consults with a QoS component to resolve resource allocation specifications instead using defaults or information encoded as part of the service "type". For example, "gold" level of Monte Carlo service type will resolve into certain priority level and 10 servers by the QoS component.

3. Quality of Services for Stateful Application Services

There are many studies in the quality of services (QoS) for batch oriented systems and for transactional processing environments. However, the QoS management of interactive parallel workload of analytic applications has not been addressed extensively.

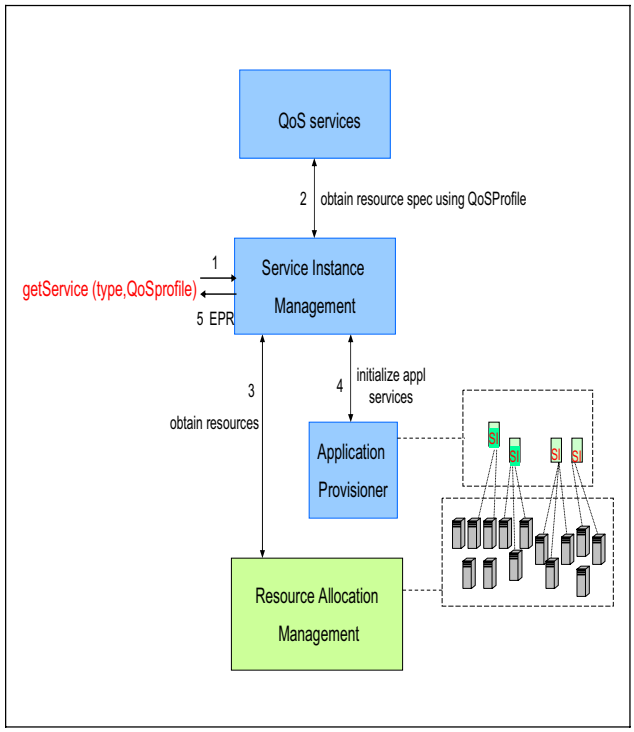

Figure 3: Interaction model of SIM

Similar to the transaction workload, there would be expected response time goals for interactive parallel works as response time would affect the productivity of interactive users. And, similar to the parallel batch jobs, the processing time for the workload has relatively high variance and may greatly depend on the level of parallelism. Liberal allocations of resources to achieve high parallel levels may provide better quality of services, but may increase total cost of system infrastructure.

The QoS management of analytic applications requires further studies by researchers. Some of the interesting topics include the following:

1. ***QoS establishment***: study what appropriate QoS specifications for application instance services are meaningful to the users, and are enforceable by the resource scheduling system (e.g. a Monte Carlo simulation service may have a response time of xx seconds as one specific type of QoS objectives).
2. ***Resource requirement based on application service profile derivation*:** enforceable QoS may require the appropriate allocation of resources hosting the service instances. One technique to explore is based on historical execution information or benchmark data. The QoS component will derive from historic or benchmark data on the required resources to instantiate a service of certain QoS objectives (e.g. Monte Carlo service ABC requires 2 servers of 1.6 GHz and 2GB of memory to achieve an average execution time of xx seconds).
3. ***Service instance pool management***: proactively creating and pre-configuring application service instances to ensure that the service activation time is short enough to meet the response time objectives; maintaining appropriate number of service instances of various application services in pools to meet the workload demand (e.g. based on the arriving rate of Monte Carlo service requests); when demands decline, superfluous service instances will be released
4. ***Resource partition management***: within an infrastructure system, resources used by analytic application workload may be shared with batch workload and/or transactional workload for simplicity of system management and achieving improved total utilization; there are challenges in managing mixed workloads with differentiating QoS objectives and with different rates of resource consumption by various workloads

As part of our research project, a service instance pool management prototype (item 3 in the above)

was developed [17]. This prototype used a cost model to study the effectiveness of varying algorithms used to manage the numbers of service instances for different applications within a computing cluster.

4. Related Work

The GrADSolve [18] project evolved from GrADS [6] and NetSolve. The NetSolve of GrADSolve supported the acquiring of grid resources to the execution parallel applications that were developed using the RPC model. The RPC model may be appropriate for developing applications to be deployed unto the service instances of our project. The project also studied some load-balancing issues. However, it did not consider analytic application workload as characterized in this paper.

MedIGrid [3] was a distributed application developed to use distributed resources managed as grid resources. It used Globus toolkit [7] for resource management.

The resource management for Globus, GRAM, neither specifically addressed the requirements of the analytic applications nor defined the support for stateful invocation.

5. Conclusion and Future Work

We extracted and provided detailed descriptions on a set of attributes and requirements from existing analytic applications in industrial segments such as capital markets and medical image processing: relatively long processing time, parallellized, multiple stateful invocations, application as web services, and with quality of services goals. To improve support for these classes of analytic applications, we proposed the stateful support services, by extending resource scheduling systems capable of running both serial and parallel jobs.

On the design and implementation of the stateful support services of getService and releaseService, we explored different approaches with various level of sophistication. We find that this is a fertile area for research and plan to pursuit further.

We also suggested the need to understand workload characterization of the analytic applications, and included some research directions for investigating issues related to quality of services for these applications. Our research plan includes building a couple of product level application services using the stateful support for empirical studies and performance evaluations.

6. Acknowledgements

We would like to acknowledge Waiman Chan, Asit Dan, Nathan Falk, Kevin Gildea, David Jensen, Alan King, Chin Lee, Gary Mincher and Kavitha Ranganathan for their contribution in our collaborative design effort.

7. References

[1] Ansorge, R. E., Carpenter, T. A. et al. “Parallel Comupting with MPI for Medical Image Codes”. http://www.wbic.cam.ac.uk/~rea1/research/MPIpaper.pdf

[2] Avramidis, T. A., Zinchenko, Y., Coleman, T. F., Verma, A. “Efficiency Improvements for Pricing American Options with a Stochastic Mesh”, Financial Engineering News, Dec. 2000

[3] Bertero, M., Bonetto, P., et al. ”MedIGrid: a Medical Imaging Application for Computational Grids”. Proceedings of IPDPS 2003, April 2003.

[4] Casanova. H., Dongarra, J. “NetSolve: A Network Server for Solving Computational Science Problems”. International Journal of Supercomputer Applications and High Performance Computing, Vol. 11, No. 3, 1997.

[5] Collins, D. L., Montagnat, J., et al. “Automated Estimation of Brain Volume in Multiple Sclerosis with BICRR”. Proceeding of the Annual Symposium on Information Processing on Medical Imaging, vol 2082, Lecture Nodes in Computer Science, Springer, 2001.

[6] Cooper, K. et al. “New Grid Scheduling and Rescheduling Methods in the GrADS Project”. 18th International Parallel and Distributed Processing Symposium (IPDPS'04) – Workshop

[7] Foster, I., Kesselman, C. "Globus: A Metacomputing Infrastructure Toolkit," The International Journal of Supercomputer Applications and High Performance Computing, vol. 11, no. 2, 1997.

[8] Germain, C., Breton, V., et al. “Grid-enabling Medical Image Analysis”. Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid (Biogrid'05), Cardiff, UK, May 2005.

[9] Glatard, T., Montagnat, J., Pennec, X. “Grid-enable Workflow for Data Intensive medical Applications”. http://www.i3s.unice.fr/~glatard/publis/CBMS05.pdf

[10] Imielinska, C., Molholt, P. “Incorporating 3D Virtual Anatomy into the medical Curriculum”. Communication of ACM, Feb. 2005. Vol 48, No 2.

[11] Leong, M.-P., Cheung, C.-C., et al. “CPE: A Parallel Library for Financial Engineering Applications”. IEEE Computer, Vol 38, No 10, October, 2005.

[12] IBM Corp. “Workload Management with LoadLeveler”. 2001. http://www.redbooks.ibm.com/redbooks/SG246038.html

[13] Platform Computing Inc. http://www.platform.com/Products/Platform.LSF.Family/
[14] Metraxas, D. “Medical Image Modeling, Tools and Applications”. Communication of ACM, Feb. 2005, Vol 48, No 2.
[15] Montagnat, J. Davila, E. Magnin, I. E. “Efficient Visualization of 3D Medical Scenes for Remote Interactive Applications”. Image and Signal Processing and Analysis, September, 2003, Roma, Italia.
[16] Porada, W. “The Changing Face of Risk Management”. http://www.midas-kapiti.com/files/file5298_Misys%20Risk%20Management%20Whitepaper.pdf
[17] Ranganathan, K. and Dan, A. "Proactive Management of Service Instance Pools for meeting Service Level Objectives“, International Conference on Service Oriented Computing ICSOC 2005
[18] Vadhiyar, S. S., Dongarra, J. J. ”GrADSolve: a grid-based RPC System for Parallel Computing with Application-level Scheduling”. Journal of Parallel and Distributed Computing”. Vol 64, Issue 6. June 2004.
[19] Waters Financial Technology Intelligence, December 1, 2003 http://www.watersonline.com/public/showPage.html?page=129214
[20] OASIS, “Web Service Resource Framework – Primer” http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf
[21] IBM Corp. “Websphere Extended Deployment Version 6.0. 2005 http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp?topic=/com.ibm.websphere.xd.doc/info/odoe_task/codoeprovision.html