Experiments on a Grid Layer Prototype for Shared Data Programming Model

Dacian Tudor*, Georgiana Macariu*, Wolfgang Schreiner** and Vladimir Cretu*
* "Politehnica" University of Timişoara, Computer Science Department, Timişoara, Romania
dacian@cs.upt.ro, georgiana@cs.upt.ro, vladimir.cretu@cs.upt.ro
** Research Institute for Symbolic Computation, Johannes Kepler University, 4040 Linz, Austria
wolfgang.schreiner@risc.uni-linz.ac.at

Abstract —**In the grid context, there is little support for programming paradigms such as shared data or associative programming. We have previously proposed an original idea to attack shared data programming on the grid by making use of both relaxed consistency models and user specified type consistency in an object oriented model. In this paper we present the experimental results of GUN, a lightweight Java prototype implementation of our shared data programming model. In order to assess GUN's performance, some generic experiments are designed. First, GUN's correctness has been proved in a real grid deployment where a limited number of nodes were available. In order to observe GUN's scalability with an increasing number of nodes, experiments have been conducted in a simulated large scale grid environment. The conducted experiments have shown a good scalability of GUN, as well as promising results in terms of response times and usability.**

I. INTRODUCTION

Distributed systems' analysis is a very important, complex and sensitive topic. One of the most common analysis domains relate to performance analysis where the performance aspects of the system are aimed to be highlighted. In many situations these aspects measure response times for various operations. A simple example is the total execution time of a distributed algorithm, given certain input data and a particular system deployment. Other aspects of distributed systems' analysis relate to the number and capacity of used resources in order to complete a certain task. Quality related analysis such as different kinds of statistical information represents another dimension of distributed system's analysis. Independently of the specific analysis that is aimed, a common problem in distributed systems and especially in grids is to be able to reproduce a given system condition such as the number of deployed machines, characteristics of the connectivity layer (e.g. bandwidth, latency, network congestion) or machine characteristics (system load, free memory, resource distribution). As one can immediately notice, the deployment of real-life scenarios in an open environment leads to a very high number of possible combinations. As a result, most system evaluations are preferred to be done in ideal conditions where for example there only running the application under test and the connectivity layer is closed to the outside world. Also, the number of the deployed machines is in most of the times quite modest.

Evaluating a grid system in ideal conditions is straightforward, but it opens the question of reproducibility likelihood, meaning that if one wants to reproduce a given experiment, one must ensure a similar environment. Due to the complexity of grid systems, sometimes this requirement cannot be achieved. Worse, a real-life experimental scenario is almost impossible to reproduce in case of a large scale distributed application deployed on a wide area grid. This brings us to the idea of considering other means of system evaluation that could give the possibility to correlate results. We aim to perform system analysis on three different directions: theoretical analysis, prototype-based analysis and computer aided analysis. In this paper we focus on prototype-based analysis.

In general, we distinguish three main domains while analyzing a grid system: performance analysis, resource related analysis and quality related analysis. In our view, performance related analysis refers mostly to the elapsed time for a given operation such as response time for a given request. Resource related analysis refers mostly to the resource usage in order to complete a given operation such as CPU or memory usage. It might also refer to resource status information like latency and bandwidth. Quality related analysis comprises statistical information that is collected during the execution time of an application in a distributed system. It may refer to operation acceptance/revocation ratio, time-based availability, throughput, correctness level etc. Generally speaking, quality related analysis is closely related to the analyzed system as it aims to highlight domain specific aspects opposite to generic aspects found in the other two categories.

II. GUN PROTOTYPE

GUN is the acronym for Grid UNiverse and represents a Java based implementation of the grid universe model defined in [1]. Remote interactions are expressed in GUN based on Java's remote object model. First, the Remote Method Invocation (RMI) solution was chosen for its simplicity and ease of use. Second, because the system model does not require multicasting support (like Jini [2] or ProActive [3] solutions do for example), the RMI model fits well to the abstract model.

GUN reflects the architecture of the abstract model and the abstract system architecture described in [1]. Similar to the abstract model, in GUN there are a set of processes deployed over several networks called universe nodes. The universe nodes are homogeneous and each of them is able to accommodate a certain number of data items, until the available capacity of the universe node is consumed. Typically universe nodes are grouped together in network latency proximity and form a universe. The collection of all deployed universes forms the grid universe. Each universe contains a dedicated node called "primary node" which manages the communication with other universes

and indexes the information on available data items accommodated by each node within the same universe. All primary nodes can be seen as a distributed registry, each being responsible for managing certain number of data objects.

The GUN prototype is divided into three layers, as illustrated in Figure 1. There is a user layer which exposes the abstractions and necessary interfaces to the application programmer. The second layer is the kernel which implements the core algorithms and implements all interfaces exposed to the outside world by the user layer. Last but not least, there is a replication layer which handles object replication policies. The replication layer implements an interface required by the kernel so that the kernel invokes the replication engine at some key points in order to trigger object replication. The replication layer is extendable, meaning that user defined replication rules can be registered into the GUN architecture.

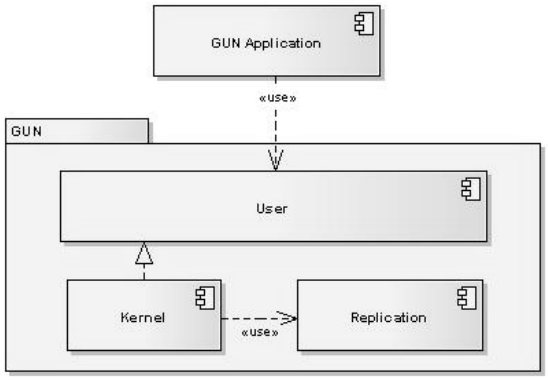

Figure 1. GUN Architecture

III. EVALUATION CRITERIA

In order to analyze the abstract model introduced in [1], we introduce several evaluation criteria as analysis metrics that shall be used while referring to different aspects of our system. We associate an abbreviation to each of the criteria in order to refer to them easily. We consider that a test application is running in the grid system on one or more nodes. The application injects different stimuli into the distributed system in order to observe and analyze certain aspects.

Most of the performance evaluation criteria for shared data programming solutions refer to the application execution time [4], [5]. In some cases like [4] resource consumption indications are presented, but generally the focus is the execution time and scalability. We define a set of measurements for each of the three evaluation categories which emphasizes both generic and specific aspects of the shared data model. GUN's kernel logs all internal events and processes primary event logs into specific higher-level metrics. Depending on the application type and the desired behaviour to be observed, a specific set of criteria can be selected for analysis. Next we summarize some of the most important measurements performed by the GUN kernel.

In the context of performance analysis, we define the following measurements which serve as a performance evaluation criteria for our model:

- CT [ms]: completion time which represents the execution time for a distributed application, from the time the application process starts execution until the process finishes.
- AQT, AQET [ms]: acquire (exclusive) time which represents the time elapsed from the moment of issuing an acquire (exclusive) request from a node until the acquire operation is granted.

In terms of quality related analysis, we define the following measurements which serve as a performance evaluation criteria for our model:

- AQSR, AQESR [%]: acquire (exclusive) success rate which represents the success rate for all issued acquire (exclusive) operations during the application execution. An acquire operation is successful if the operation is granted within the demanded timeout.

IV. PROTOTYPE EXPERIMENTS

This section describes different experiments that aim to highlight performance, resource aspects as well as qualitative aspects of the abstract model. Evaluation of a generic application is most of the times a very hard task due to the lack of information about the application external stimuli which basically translates into the orchestration of distributed component interaction. In other words it is hard to devise a general behavior rule based on unknown interactions (both data and service). As a result, opposite to evaluating a concrete application, we aim to evaluate different interaction patterns that can be interactions within a real-life application. Any application can be decomposed into a set of such interactions patterns.

We consider that we have a number of m universes deployed. Each of the m universes contains a number of n nodes. Each of the n nodes has a capacity c, thus there is a homogeneous node distribution across universes. Depending on the experiment, at a given time a number of p processes are running in the grid universe where $p <= m \times n$. An experiment defines a concrete interaction pattern which focuses on one or more performance aspects of the global system and sets the experiment frame. An experiment variant defines the exploration in the system parameters space where different parameter configurations are used.

A. Grid Object Search

The purpose of this experiment is to evaluate the characteristics of the search operation in the grid universe. The experiment consists of the following steps:

- [S1 - Deployment] A number of $p = m \times n$ processes are deployed, one process on every grid node within the grid universe.
- [S2 - Creation] One of the p processes creates a number of o grid objects in the grid universe within a range of numerical object identifiers [OID1, OID2].
- [S3 - Steady state] Each process out of the p processes issues a search request in the domain of the object identifiers, in order that objects are eventually replicated and a steady state is reached.
- [S4 - Measurement] Each process performs a random search operation in the domain of the object identifiers that have been created and the evaluation criteria are logged.

The experiment has the following variants:

- [V1 – Replication] The following replication policies shall be used: "one object per universe", "n/2 objects per universe" and "one object per node".
- [V2 – Object Type Count] The number of different object types that are created in S2 should be {1, 10, 100, 500, 1000}.
- [V3 – Node count] The number of nodes within universes should be {5, 10, 20, 50}.

The expected result was that a decrease in search time is expected in case where object replication is enabled. Second, the search time should not be directly dependent on the number of nodes and number of objects.

B. Acquire Corectness

The purpose of this experiment is to observe the correctness of the acquire operation and evaluate its performance under various conditions. The experiment consists of the following steps:

- [S1 - Deployment] A number of $p = m \times n$ processes are deployed, one on every grid node within the grid universe. "One object per node" replication rule shall be used in order to maximize interaction patterns between nodes.
- [S2 - Creation] One of the p processes creates a generic grid object in the grid universe.
- [S3 - Steady state] Each process out of the p processes issues a search operation in order to trigger object replication and to reach a steady system state.
- [S4 - Measurement] Each process performs a search operation following by a number of 100 acquire requests issued with a delay of d ms. All evaluation criteria are logged.

The experiment has the following variants:

- [V1 – Node count] The number of nodes within universes should be {5, 10, 20, 50}.
- [V2 – Acquire Operations] The delay between subsequent operations shall be {3000ms, 2000ms, 1000ms, 500ms, 100ms}.
- [V3 – Client count] The number of client applications is limited to one and the number of nodes is the maximum possible number.

The expected results were that a 100% acquire success rate shall be noticed independent on the experiment's variants.

C. Acquire Exclusive Corectness

The purpose of this experiment is to observe the correctness of the acquire-exclusive operation and evaluate its performance under various conditions. The experiment consists of the following steps:

- [S1 - Deployment] Same as Acquire Correctness case.
- [S2 - Creation] One of the p processes creates a generic grid object in the grid universe with the following data: a user provided data content (string) and a version (number) that is incremented automatically each time the content is set.
- [S3 - Steady state] Same as Acquire Correctness case.
- [S4 - Measurement] Each process performs a search operation following by a series of 100 acquire-exclusive requests issued with a delay of d ms. All evaluation criteria are logged. After each acquire-exclusive operation, each process shall log the data content and the version of the object is operates on.

The experiment has the following variants:

- [V1 – Node count] The number of nodes within universes should be {5, 10, 20, 50}.
- [V2 – Acquire Operations] The delay between subsequent operations shall be {3000ms, 2000ms, 1000ms, 500ms, 100ms}.
- [V3 – Client count] The number of client applications is limited to one and the number of nodes is the maximum possible number.

The expected results were that in all logged grid object sequences, there should be no entries belonging to different processes that have logged the same object version (acquire exclusive correctness criteria).

V. EXPERIMENTAL RESULTS

The GUN prototype was tested in a large scale grid environment which is depicted in Figure 2. During the prototype development and testing phases, a number of 3 to 5 universes were deployed. Universes were made out of machines belonging to networks located at "Politehnica" University of Timisoara, Western University of Timisoara, Research Institute for Symbolic Computation (RISC) in Linz, Austria, "Technische Universität Dresden" and the Bangalore Institute of Technology University. In this configuration, the latencies between universes were ranging between 10ms and 180ms.

One of the first problems encountered during the execution of the first experiments was that the number of required nodes was higher than the number of available machines. Most of the clusters that were accessed had between 3 and 10 machines and the experiments require a number of nodes ranging from 5 to 50 (in the ideal case). The problem was exacerbated because there were no common access interfaces to those networks so that a significant effort was required to deploy GUN and start the experimental applications automatically. As a result, another experimental setup has been investigated which should provide a much higher number of nodes in order to observe high level scalability aspects.

For this purpose, a SGI Altix 4700 machine was used, which is available at the Research Institute for Symbolic Computation (RISC) in Linz, Austria. The machine located at RISC has 128 Intel Itanium 2 Montecito processors with hyper-threading technology, running at 1,6GHz and having 18 MB L3 cache which means that it can execute 256 threads simultaneously. The machine has 1 Terabytes of main memory (Global Shared Memory NUMA) and 24300 GB data storage.

The second problem appeared due to the Java runtime environment which was provided by the JRockit java distribution which is supposed to be optimized for the Altix machines and the Linux SUSE operating system. Preliminary tests using the JRockit solution showed a non-scalable behavior of the GUN prototype although the theoretical analysis highlighted the contrary situation. The reasons for this unexpected behavior were identified during investigations which have been conducted for almost one month time, which have showed that the JRockit runtime environment is not scalable in terms of

the RMI server implementation. In other words, the RMI server does not scale to a high number of clients and therefore introduced a bottleneck in our system. This means that the lack of scalability was due to the Java virtual machine implementation for the Altix machine. The problem was fixed by using an adapted version of SUN's JDK version 1.6 for 64 bit machines which does not have the same scalability problem. Using the new JDK, GUN's behavior was according to the initial expectations.

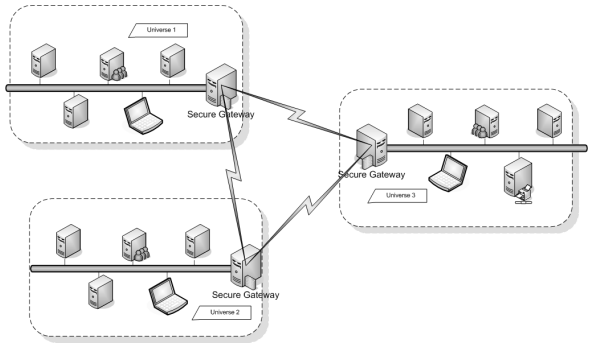

Figure 2. GUN Experiments Setup

The environment problems were not completely overcome because the latency between universes was not present anymore in a natural way. As a result, artificial delays between remote calls were introduced in the GUN prototype in order to reflect a real deployment. In order to reduce the risk of uncontrolled thread scheduling, a spinning wait was used. A latency of 10ms was considered for calls within a universe and 50ms for calls from one universe to another. It is important to note that these values are highly dependent on the remote method signature as well as their values (e.g. in case of list of various objects), as all method parameters are serialized and transferred over the network. This aspect is not fully covered in the experiments running on Altix as it naturally happens in a real deployment. However, the remote execution penalty in the real wide scale distributed environment was not higher than 250ms, considering the parameters for all remote methods defined in GUN. In case of only the European clusters, the latency was between 40ms and 90ms. As a result, the fixed value of 50ms was considered in the Altix evaluation setup.

A. *Grid Object Search*

In case that "one object per node" replication rule is used, the search time dependency to the number of objects, ranging from 1 to 1000 is shown in Figure 3 for a configuration of 30 nodes per universe. The first group of client nodes is the ones belonging to the universe where the initial object has been created. Due to the replication policy, an object replica exist in all universes, thus the same search time is experienced by all applications independent on their universe membership.

Similarly, the same results can be observed in Figure 4 and Figure 5, where "n/2 object replication" and "one object per universe" replication rules are used. In all previous situations one can notice a uniform distribution of search time values independent on the number of grid objects. The search time oscillations can be accounted on higher system load and unpredictable operations of the java run-time environment (e.g. garbage collection etc).

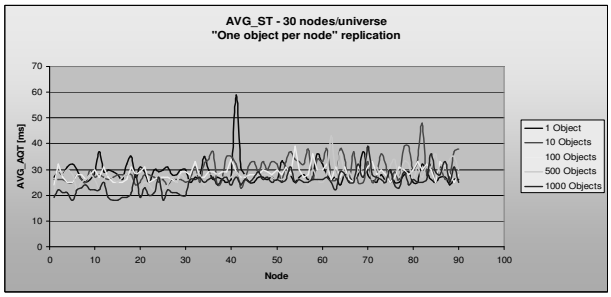

Figure 3. Grid Object Search – One object per node

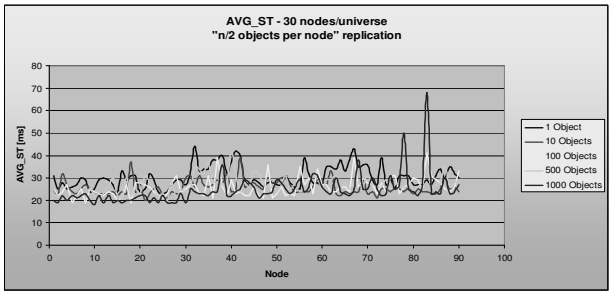

Figure 4. Grid Object Search – n/2 objects per node

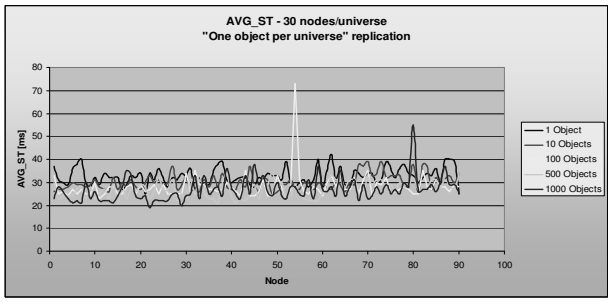

Figure 5. Grid Object Search – One object per universe

In case no replication is used, the search time dependency to the number of objects, ranging from 1 to 1000 is shown in Figure 6 for a configuration of 30 nodes per universe. The first group of 30 client nodes belongs to the universe where the object was created. As a result, the clients residing in that universe have small search time values. The next group of 30 clients is located in a different universe and the search request if forwarded across the universe registration graph. In this case the first contacted universe during the search operation has the object. As a consequence, the search times are higher with approximately 100ms (round trip request). The last group of 30 nodes belongs to a different universe. The search requests are first directed to the second universe's primary node and then to the first universe where the object resides. Thus, the search time is approximately twice the value as in the later case.

The experiment results with different number of nodes within a universe, in case of 1000 grid objects where one object per node and no replication is used are presented in Figure 7. It can be noticed that the search time does not depend on the number of nodes. In case where no replication is used, the same pattern described in Figure 6 occurs: the search time increases together with the search chain length. For every search indirection, the search time increases by one round-trip inter-universe latency (approximately 100ms).

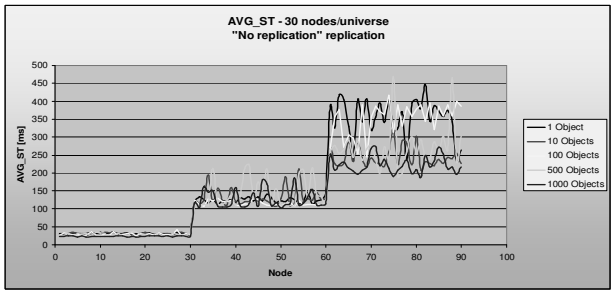

Figure 6. Grid Object Search – No replication

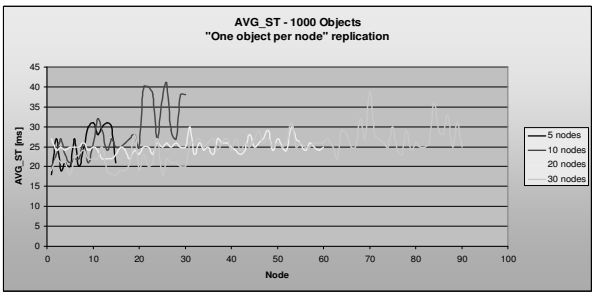

Figure 7. Grid Object Search – node number dependency

B. Acquire Corectness

It is important to note that in all experiments that were executed, AQSR = 100%, meaning that all acquire requests were successfully satisfied within the chosen timeout of 3000ms. For all node configurations different experiment parameters (d) have been used. Figure 8 and Figure 9 shows the average acquire time (AVG_AQT) and the average release time (AVG_REL) values in 5 different situations where the waiting time d is set to 100ms, 500ms, 1000ms, 2000ms and 3000ms. The first group of nodes has in some cases significantly smaller time values belong to the universe where the token is. In case of the acquire operations, if the token is not owned by the universe where the requestor resides, the token is asked for acquire permission. This request translates to a remote request and remote callback which in our case adds about 100ms. Besides the remote round trip call penalty, one needs to consider processing time in the primary node queue which is dependent on the number of requests.

As seen in the diagrams, in case the acquire operations are issued with a delay of 2000ms respectively 3000ms, GUN shows a good and stable performance independent on the node location. There is only a slight increase in acquire time for the nodes belonging to the universes that do not have the token. This is quite normal since the acquire request has to pass the universe boundaries (remote call over large latency connection). In the other three cases, here is a performance degradation in the system when the acquire requests are issued more rapidly. This happens because all nodes are issuing request towards the primary node that holds the token and the requests are serialized in a queue. If the requests frequency is higher than the processing frequency, the requests are accumulating in the queue and the waiting time increases. It is worth to note that the processing frequency depends on the inter-universe network latency because the responses are sent via a large latency connection. Contrary to the acquire time, the release time remains stable independent on the request frequency (acquire frequency is equal to the release frequency). Only in the extreme case where requests are issued within each 100ms, a variation of the release time can be noticed but this can be accounted to the global task scheduling mechanism.

The performance results for average acquire time if different nodes and request frequencies were used are shown in Figure 10 and Figure 11. It can be observed that the same pattern described above appears in all node configuration with the remark that the impact on the acquire time is direct proportional to the number of nodes. This is quite normal since the number of requests depends on the number of nodes.

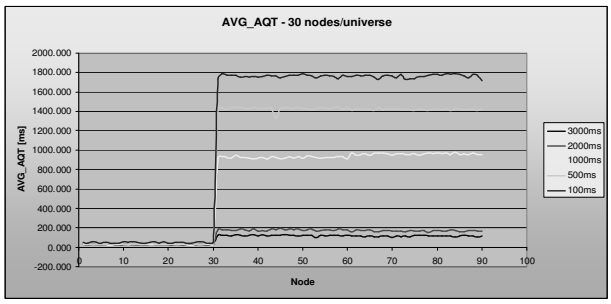

Figure 8. AVG_AQT for 30 nodes per universe

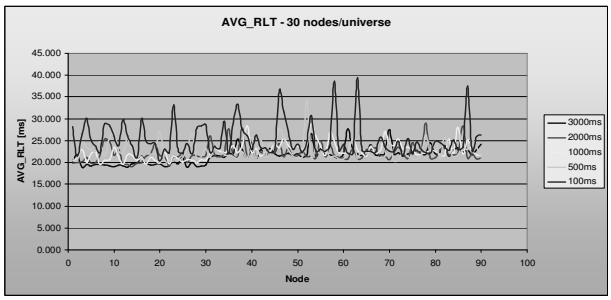

Figure 9. AVG_RLT for 30 nodes per universe

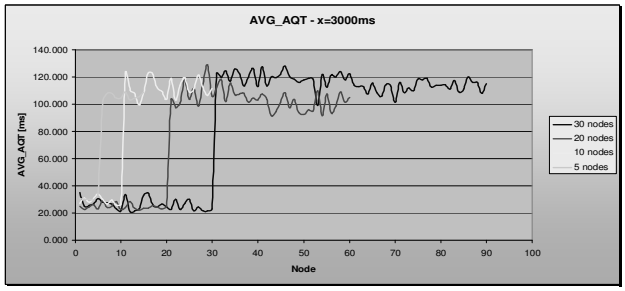

Figure 10. AVG_AQT for 3000ms and different nodes

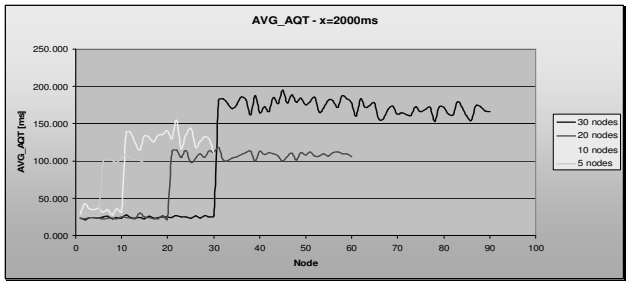

Figure 11. AVG_AQT for 2000ms and different nodes

C. Acquire Exclusive Corectness

In this experiment a "one object per node" replication rule has been used in order to favor higher interaction patterns and put more stress to the GUN prototype. It is important to note that in all experiments, the number of acquire misses did not exceed 1%. Using a proper timeout value, AQESR = 100%, meaning that all acquire exclusive requests were successfully satisfied. In case of this experiment the timeout value was set to 20000ms. Second, during all the experiments, the acquire exclusive correctness criteria that in all logged object value entries, there should not be entries belonging to different processes that have logged the same object value, hold true.

For different node configurations, different experiment parameters (d and x) have been used. Figure 12 and Figure 13 show AVG_AQET and AVG_RLT value in 8 different situations where the waiting time d is set from 50ms to 5000ms. It can be observed that acquire exclusive time remains constant up to a value of the wait time between delays (d). Below a given value, which in our case is about 500ms, the system starts to exhibit a degraded behavior because of the accumulated requests in the primary node's queue.

In case of the release time, it remains constant independent on the request frequencies, because of an asynchronous mechanism built into the release request. Basically, the release request is delegated and handled in the primary node such as the caller does not have to wait until it is processed.

The situation where different number of nodes has been configured is represented in Figure 14 and Figure 15. If the delays between subsequent requests are relatively high (e.g. 5000ms), there is no significant difference in terms of the acquire exclusive time. As the delay decreases, the experiments have shown a difference (as in Figure 15) which is caused by the number of the acquire requests. The number of the acquire requests is directly dependent on the number of nodes.

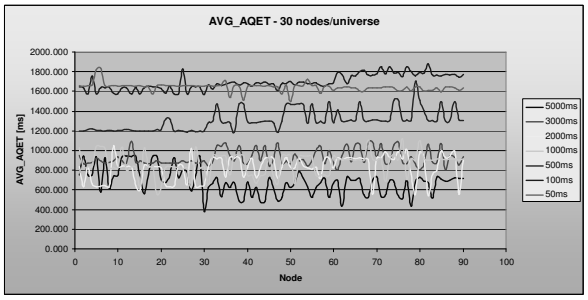

Figure 12. Acquire exclusive time

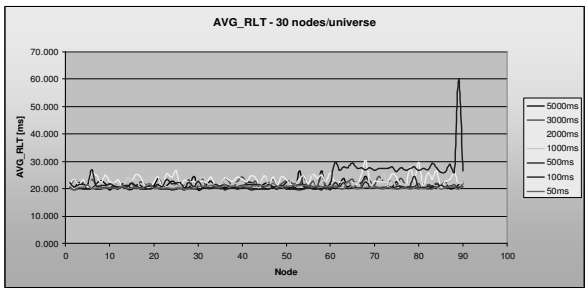

Figure 13. Acquire exclusive release time

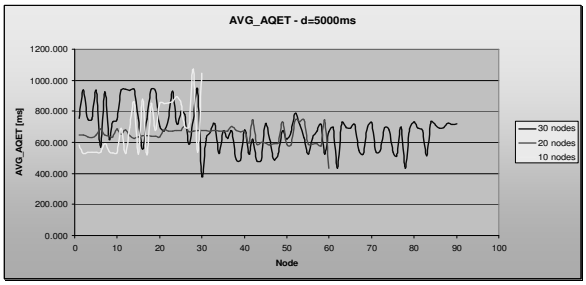

Figure 14. Acquire exclusive node number dependency

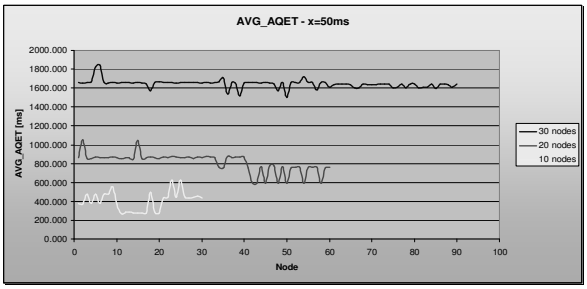

Figure 15. Acquire exclusive node number dependency

VI. CONCLUSIONS

In this paper we have presented the experiments we have conducted on a Java based prototype for shared data programming model. We have started by defining the context for the experiments and their workflow, measurements and expected results. Next we have presented a summary of the core experiments we have conducted and their results. The results have shown a good scalability in the number of deployed nodes and a good performance in terms of execution time.

REFERENCES

[1] Dacian Tudor, Vladimir Cretu and Wolfgang Schreiner, "Designing an Architecture for Distributed Shared Data on the Grid", Proceedings of the International Conference on Algorithms and Architectures (ICA3PP-2008), A. Bourgeois and S.Q. Zheng (Eds.), LNCS 5022, pp. 261-264, Springer-Verlag Berlin Heidelberg 2008.

[2] Baker, M. Smith, G., "Jini meets the Grid", Proceedings of the International Conference on Parallel ProcessingWorkshops, pp. 193-198, ISBN: 0-7695-1260-7, 2001.

[3] Baduel, L., Baude, F. and Caromel, D., "Efficient, Flexible and Typed Group Communications for Java", Joint ACM Java Grande - ISCOPE 2002 Conference, Seattle, Washington, pp. 28-36, ISBN 1-58113-559-8, 2002.

[4] Gabriel Antoniu, Luc Bouge, and Mathieu Jan. JuxMem: An adaptive supportive platform for data sharing on the grid. Scalable Computing: Practice and Experience, 6(3):45-55, November 2005.

[5] Julien Sopena, Fabrice Legond-Aubry, Luciana Arantes, Pierre Sens. A Composition Approach to Mutual Exclusion Algorithms for Grid Applications. Proceedings of the 2007 International Conference on Parallel Processing (ICPP 2007), Volume 00, Page: 65, ISBN 0-7695-2933-X, 2007