

E1 Project Title

Towards High-performance and Fault-tolerant Distributed Java Implementations

Summary:

Java Virtual Machines form an important part of the web and business server market. Distributed Java Virtual Machines have the potential to make a significant contribution to industries that utilize this technology. An attractive platform for this purpose is the *cluster*, a highly cost-effective and scalable parallel computer model. However, realizing on such a platform a high performance virtual machine implementation tolerant to hardware and software faults, and having efficient memory utilization, presents many challenging research issues. This project will address these issues by extending a highly efficient and extensible Java implementation to be aware of its cluster environment.

E2 Background

The market for high performance computers has polarized, with SMP systems at the high end and Beowulf style cluster multicomputers, constructed from commodity components and providing enormous aggregate performance for comparatively low cost, at the other. In the applications domain, such computers are increasingly seen as networked high-performance compute and data servers, and the problem set now includes large-scale problems in information management and retrieval, multimedia repositories, data warehousing, data mining, and database transaction processing, as well as the more traditional computational science and engineering problems.

Over recent years, the programming language Java has enjoyed increasing popularity in the above application areas. Key to this a Java Virtual Machine (JVM) that provides a well defined abstract machine. This abstraction, combined with dynamic compilation, can make efficient use of the underlying hardware (actual machine). With its virtually universal vendor support, this also permits Java applications to enjoy a very high level of portability. Java also provides a convenient and powerful object-oriented programming model. As a result, efficient and reliable implementation of Java on high performance computer platforms has become an important focus of computer science research. Many of these applications, such as server applications, exhibit loose coupling between the threads. The resulting tradeoff between computation and communication makes the use of a cluster of workstations feasible.

This project will develop a high performance, reliable, distributed JVM that is cluster aware, and yet hides the distributed nature of the hardware from the application layer, providing the infrastructure for implementing and analyzing distribution techniques and runtime support algorithms for multi-threaded applications written in Java, typical examples being web servers, video servers and cluster gateway control software [9]. Such applications demand high performance with respect to both transaction throughput and response time, be scalable with respect to database size and transaction rates, and be able to run for long periods without interruption. Large-scale clusters are thus required to meet high transaction rates. With the requirement to be long-running, the (virtual) machine executing the application must therefore be itself scalable, and be tolerant to hardware and software faults.

The project brings together researchers with expertise in Java implementation, parallel compilers, distributed object stores, algorithms, garbage collection and cluster computers, in order to

research the many issues required to realize an efficient, fault-tolerant distributed JVM. It also builds upon the successes of previous projects which these researchers have been involved with: The Platypus Distributed Object Store[14] (ANU), the Bunyip Large Cluster Computer [25] (ANU), and the Beltway Garbage Collector Framework[7] (University of Massachussets), not to mention a prototype version of a distributed JVM (ANU) [26].

The Jikes RVM approach All current work on building a distributed JVM is based on modifying an existing JVM to work in a distributed environment. This project takes a similar approach, but differs from the rest in using the Jikes RVM as the base JVM.

The Jikes RVM [4] was developed by IBM Watson Laboratories and is the first JVM written in Java¹. The Jikes RVM has been designed for the purpose of research, with its source codes being publically available under the Common Public Licence, and has an extensible framework ideal for research in JVM-related issues. The framework supports code introspection, performance monitoring and sophisticated garbage collection strategies. It achieves high performance through its optimizing compiler and its use of lightweight threads with low Operating System overheads, and is competitive with commercially available JVMs [4]. Being written in Java itself facilitates more rapid development of extension codes. The Jikes RVM has research teams actively engaged in JVM-related research, including those from several IBM labs, UMass, etc. (see <http://www-124.ibm.com/developerworks/oss/jikesrvm/info/users.shtml>). It thus forms an ideal basis for this project.

Background on our distributed JVM (dJVM) There are many research projects looking at effective techniques for developing single system image (SSI) cluster enabled implementation of a JVM. The approaches taken can be broadly divided into three categories: implementation above the VM using either static [10] or dynamic transformation of Java classes; building the VM on cluster enabled infrastructure [19]; or building a cluster aware VM [2]. We see the last of these, combined with dynamic byte-code transformation techniques [21] both at build-time and runtime, as the most promising and flexible approach.

An initial investigation into distributed SSI JVMs, under the CAP Fujitsu research program, resulted in the development of a prototype within a single year [26]. The prototype is capable of running pure Java programs on a cluster of commodity Intel Linux machines, including a large Beowulf cluster [25]. Distributed computation is effected by executing code based on object location [2], and thus relies on remote creation of objects for distribution. To support this model, the prototype incorporates: the masquerading of local thread resources as global threads; remote acquisition and ownership of classes; implementation of a local and remote referencing scheme; and the use of class and code transformation techniques combined with modifications to the non-optimizing compiler to facilitate remote object access. The prototype is far from complete, but provides a proof of concept of our approach.

The prototype is thus an initial step providing some infrastructure for conducting research into an SSI JVM implementation. Critically, the prototype does not address: object and thread migration, caching or object placement based on code analysis or runtime heuristics; fault tolerant operations; distributed garbage collection; and the full exploitation of dynamic transformation techniques for implementing semantic and functional changes in the runtime system and applications. These are all areas fundamental to the effective execution of Java programs in a SSI cluster aware JVM.

¹Some C code is used for boot strapping and as glue between the VM and a small number of C library functions.

E3 Significance and Innovation

As far as we are aware, our prototype is the first (and only) distributed implementation of a JVM written entirely in Java. This project is unique in that it simultaneously addresses the three issues of performance, distributed garbage collection and reliability (fault-tolerance). All of these issues interact non-trivially with each other.

As all of these aspects will be integrated into a living system capable of running realistic applications, the resulting dJVM for this project will yield performance evaluations of practical significance on these issues. This, we believe, is lacking in the literature, where evaluations are largely based on theoretical considerations or simulation studies. As well, to our knowledge, very little work has been done on distributed garbage collection and reliability issues in the context of a distributed JVM.

Key to the success of an SSI distributed JVM is good performance, which necessitates the effective use of the underlying hardware. This requires the computational load to be distributed while minimizing the impact of communication overhead. Although, techniques for placement, migration and caching have been the subject of investigation [2, 19], much work remains to address the tension between load distribution and communication overhead, particularly in the context of a reliable and fault tolerant distributed JVM. In addition to the extension of these techniques, we see further opportunities that utilize both clustering/declustering techniques to improve temporal locality of data and automatic parallelizing compiler techniques to exploit the parallelism inherent in many server applications.

While there is much work in the literature on fault detection and fault recovery mechanisms, these issues are largely considered separately. Our project will integrate these mechanisms, and consider the tradeoffs between the degree of fault tolerance and its overhead in terms of time and memory, both during normal operation and in the presence of faults. The re-introduction of restored machines (nodes) back into the dJVM is another example of an important practical issue largely neglected in the literature; our project will address this. We will also be in a position to explore the further issue of which mechanisms are best implemented in the high-level system (in this case the dJVM itself), and which mechanisms are best implemented in lower-level layers (e.g. the underlying communication protocol).

In the case of distributed garbage collection (DGC), there are very few studies on how the DGC algorithms perform in a real working system, such as a distributed JVM. The concept of clustering of objects [15] has yet to be applied to an object-based Single System Image system on a cluster. We see opportunities here for improving performance (better data locality) and the incorporation of such a technique into GC algorithms.

Furthermore, in the synergy between GC and fault-tolerance mechanisms, we see interesting and unexplored opportunities and challenges. For example, a scan of the object space by the JVM could be used simultaneously for checkpointing (a mechanism used for fault recovery) and GC. If, on the other hand, object replication is used instead, then the GC mechanisms must be modified to deal with this accordingly.

This project will thus be doing ground-breaking work in the above areas. It is also the only distributed JVM project in Australia. All of the code will be made available under CPL.

Our target machine, a 192 CPU Beowulf cluster, is an award winning high performance machine. The completed system will provide an excellent platform for testing and doing research work on distributed Java. It is also substantially larger than the clusters used in the performance evaluations of distributed JVM implementations, and will thus permit better tests of scalability than has

been accomplished before. It will also have commercial implications as a comparatively low-cost, high performance, platform for running high-throughput multi-threaded Java applications.

E4 Approach

As stated earlier, a prototype dJVM has been developed. This project will build on this prototype to make it a high performance, reliable, distributed JVM tailored for multi-threaded Java applications that have loose coupling between the threads. *The project consists of the following three inter-related themes:*

The *Performance* theme addresses the complex goal of reducing the overheads of distribution, with the particular goal of permitting scalability to large-scale clusters. This will look at improving performance by looking at issues of object placement and migration, caching and thread migration.

The *Reliability* theme addresses the need for the JVM to support long-running high-availability applications, a large component of which is the need to be fault-tolerant (for hardware and software faults). It interacts with the Performance themes in that any such mechanism should itself have low overhead and be scalable.

The *Distributed Garbage Collection* (DGC) theme supports the other two. Accurately identifying and reclaiming memory containing objects that are no longer used is essential for large, long-running applications. DGC must also be efficient, both in terms of the often conflicting goals of permitting rapid response times while maintaining high processing throughput, and in the distributed context, must also be scalable and fault tolerant.

E4.1 Performance

A distributed JVM must be scalable and exhibit good performance characteristics, meeting this goal requires the tension between the distribution of computational load and the resulting communication overhead be addressed. Consequently, the techniques for determining where computations are to be performed and for improving data locality must be considered jointly with methods of hiding and minimizing communication overhead.

In order to achieve effective load distribution it is necessary to intelligently distribute the work across the nodes in the system. This involves the creation of threads [2, 19], the migration of threads [19] and, potentially, the distribution of intra-thread computation. Data flow analysis, data locality, execution history and load information can improve node selection for thread creation and migration. Furthermore, opportunities exist for incorporating the distribution of intra-thread computation utilizing automatic parallelization techniques.

The communication overhead resulting from the disjunction of object and computation locality can significantly impede performance. This overhead is reduced by techniques such as: object placement and migration; caching; replication; and prefetching. Object placement and migration decisions benefit from techniques such as code analysis for common design patterns [2], interprocedural data flow analysis and escape analysis [12], and also from the approaches taken to clustering [17, 16] and declustering using dynamic and static profiling information. Although work has been carried out on caching and replication in distributed [2, 19] and non-distributed JVMs [11], there is still scope for exploiting the Java Memory Model (JMM). Furthermore, opportunities exist for introducing prefetching using code transformation techniques. We will investigate and extend these and other techniques for managing locality and hiding latency.

In order to effectively evaluate these techniques and algorithms, it is necessary to utilize a range of benchmarks. However, server applications normally exhibit a variety of traversal and computation patterns during their execution. To address this, we intend to construct real server applications, e.g. an historical stock market information system and the CSIRO Sentinel hot spot map system. The Platypus distributed store will be used to load objects to/from the dJVM. Performance analysis techniques will be used to identify memory and I/O bottlenecks. This will be aided by the performance instrumentation infrastructure already built into the Jikes RVM.

Dr Sankaranarayana will lead the work on object placement and migration, caching and thread migration, working closely with Mr Zigman. Dr Blackburn, Dr Strazdins and Mr He will also contribute to this work. A PhD topic is *Object placement and migration in a distributed, cluster aware, Java environment*. The focus of this topic will be to explore existing techniques in object placement/migration and caching in the dJVM environment, as well as introduce new techniques, and evaluate the impact on the overall performance of the dJVM. Dr Strazdins will also work with the student on the development and performance evaluation of the benchmarks. This work will evaluate the practical benefits of the techniques studied.

E4.2 Reliability

The second theme, the investigation of reliability in the context of a distributed JVM, will be lead by Dr Strazdins and Mr Zigman with contributions from Dr Blackburn. Fault tolerance – the graceful degradation of performance in the face of failures – is a desirable or essential feature of many business and government services, e.g. the CSIRO Sentinel hotspot map system. A fault tolerant distributed JVM has the potential to utilize commodity hardware for improved performance and reliability. Supporting fault tolerance requires a range of issues to be addressed including: reliability, failure avoidance and failure recovery.

Reliability of the underlying infrastructure, both hardware and software, removes some of that burden from the upper layers. This may trade-off both recovery and execution performance with elegance. Improving the infrastructure reliability necessitates the determination of failures while minimizing false positives (possibly due to high network or node loads). Network failures may be managed at a low level having redundant paths between nodes. Detecting and compensating for network failures [9] at that level can be implemented within or below a distributed JVM, trading-off flexibility for elegance.

The ‘heartbeat’ technique [9] is a promising approach for detecting machine failures, and this could be relatively easily incorporated within the well-separated communication layer of our dJVM [26]. This method could be generalized to estimate a measure of machine ‘health’ (which could be affected by load from external sources), and so be integrated into the object and thread placement strategies of the Performance Theme. It can be extended to detect when previously off-line nodes become available to be re-introduced into the dJVM.

Machine failures are more dramatic, requiring computation to be continued on the remaining nodes by: recovering a globally consistent state using checkpointing or transaction mechanisms; or ensuring continued operation by replicating data and computation.

Both checkpointing models characterized by Manivannan, et al. [13] and transactions models have been and continue to be explored in the literature. The application of these techniques within the context of a distributed JVM raises the important issue of *to what extent the JMM can be exploited to reduce the cost of global state recovery*. Several approaches can be explored:

- Applying independent checkpointing algorithms in the dJVM, including variants of those

in [24], and exploring the core of PJama [3] (non-distributed persistent JVM) checkpointing system.

- Refining the chain-and-spawn transaction model proposed in [8], exposing it to the application through libraries, or automating its use at code synchronization points in the Java application code and dJVM code. This transaction model provides a transactional scope in which all computation and data break a single (logically) atomic transaction into transaction chains.
- Assoc. Prof. Hosking has developed an optimistic transaction model that complies with the JMM. This model, although designed for a different purpose and context, could be exploited in a distributed setting to hide latency and to allow redundancy through replicated computation.

The transaction models provide a significant and promising area of investigation.

A PhD student will work on a topic entitled *A Fault Tolerant Framework for the Distributed Java Virtual Machine*. Initially, this will entail studying JMM and its flaws [23, 20] and examining the various coherency protocols that can be adapted to fully exploit the JMM including failure detection, check-pointing, transactions, data replication, and computation replication, with a view to applying one or a combination of these protocols to the Distributed JVM.

So far, little research has been done regarding the applicability of these protocols to the JVM environment. As there are differences between the Distributed JVM environment, and the more general form of a distributed environment for which the protocols were designed, it is expected that existing protocols can be adapted, relaxing some of their requirements. Such changes may result in a significant reduction in the amount of overhead necessary to make the dJVM fault tolerant.

An example is the JMM, which does permit a degree of incoherency within a running Java application. At a relatively small cost, operations used to manipulate objects in the JVM's memory could be extended to include updates to a change log. Such a log could then be used to construct checkpoints required by the coherency protocols.

E4.3 Distributed Garbage Collection

Garbage collection (GC) is fundamental part of the JVM runtime system and is essential for the viability of long-running Java applications. This necessity extends to the support of a distributed JVM and this theme aims to design and develop new, efficient, safe and complete algorithms for DGC. Key to this problem is the safe and complete analysis of a graph with an incoherent view of global states. In performing that analysis, information has to be exchanged between nodes, which impacts upon the network performance and introduces reliability issues. The dJVM presents a rare opportunity for pursuing this aim where the evaluation of those algorithms can be done using benchmarks, such as those developed by the Performance theme, that are not distribution aware. This theme focuses on two areas to realize this aim: developing efficient, structured approaches to the definition and construction of safe and complete DGCs; and the evaluation of new DGC algorithms against a range of existing algorithms.

Distributed Garbage Collection Framework There are several DGC algorithms described in the literature that are safe and complete [18]. A framework for combining a distributed termination algorithm and a centralized GC collector (CGC) to derive a DGC algorithm is presented in Blackburn, et al. [6], enabling the work in CGC to be explored in a distributed runtime system. One promising example for a low mutation rate distributed older generation is the use of a distributed variant of the concurrent reference counting algorithm [5].

Norcross, et al. [22] build on Blackburn, et al. [6], separating local intra-node GC from global DGC by defining in-transit messages as roots for the DGC algorithm, but further investigation of more general solutions and frameworks is warranted. In particular, a theoretic framework for hierarchical GC, developed by Mr Zigman, combined with practical approaches to CGC, such as JMTk, the Java Memory tool kit for the recent release of the Jikes RVM, developed by Dr Blackburn, Dr Cheng and Dr McKinley, present an opportunity to develop an automated approach to the development of DGC algorithms. By defining a range of mechanisms and a set of requirements for the interaction of multi-level GC algorithms, high level specification of GC algorithms, which in turn result in safe and complete GC and DGC, is possible.

The work outlined above is a fundamental part of the project and will be driven by Dr Blackburn (CI) and Mr Zigman working closely with Dr Sankaranarayana. A PhD studentship will target a longer term project titled *Automated Generation of Hierarchical Garbage Collectors*. Initially, the objective will be to investigate the application of a number of distributed garbage collectors combined with local intra-node garbage collection. The investigation of this area will provide the a solid grounding in GC, a fundamental area of modern runtime systems. The second stage will be to combine mechanisms and hierarchical models of GC algorithms to produce an automated system for GC generation. Such a system will allow the rapid development of a range of hierarchical DGC algorithms that can be tested in the dJVM.

Performance Performance characteristics of hybrid GC algorithms, and particularly hybrid DGC algorithms, become difficult to predict and experimentation or simulation is necessary. In general, pause time and throughput are the key requirements of garbage collection—high availability demands low latency and therefore low pause times, whereas nominal high computation rates demand high throughput. Both these factors are impacted upon by the communication behaviour of DGC [1]. The dJVM provides a rare opportunity for performing experiments using an application on a distributed runtime system. Mr Zigman, Dr Blackburn and Dr Strazdins will evaluate the performance of various algorithms, and Dr Strazdins will pay particular attention to the synergies and interaction between GC and fault tolerance mechanisms.

References

- [1] Saleh E. Abdullahi. *Empirical Studies of Distributed Garbage Collection*. PhD thesis, Queen Mary and Westfield College, December 1995.
- [2] Yariv Aridor, Michael Factor, and Avi Teperman. Implementing Java on Clusters. In *Euro-Par 2001, LCNS 2150*, pages 722–732, Rhodes Greece, 2001. Springer-Verlag.
- [3] Malcolm P. Atkinson, Laurent Daynes, Mick J. Jordan, Tony Printezis, and Susan Spence. An Orthogonally Persistent Java. Number 1. ACM Press, March 1996.
- [4] B. Alpern et al. The jalapeño virtual machine. *IBM System Journal*, 39(1):211–238, February 2000.
- [5] David F. Bacon, Perry Cheng, and V. T. Rajan. A Real-time Garbage Collector with Low Overhead and Consistent Utilization. In *Conference Record of the Thirtieth ACM Symposium on Principles of Programming Languages*, New Orleans, Louisiana, January 2003. ACM.
- [6] Stephen M Blackburn, Richard L Hudson, Ron Morrison, J Eliot B Moss, David S Munro, and John Zigman. Starting with Termination: A Methodology for Building Distributed Garbage Collection Algorithms. In *Proceedings of ACSC*, pages 20–29, 2001.

- [7] Stephen M. Blackburn, Richard Jones, Kathryn S. McKinley, and J. Eliot B. Moss. Beltway: Getting Around Garbage Collection Gridlock. In *Proceedings of SIGPLAN 2002 Conference on Programming Languages Design and Implementation*, ACM SIGPLAN Notices, pages 153–164, Berlin, June 2002. ACM Press.
- [8] Stephen M Blackburn and John N Zigman. Concurrency—The fly in the ointment? In Ronald Morrison, Mick Jordan, and Malcolm Atkinson, editors, *Advances in Persistent Object Systems: Third International Workshop on Persistence and Java September 1–3, 1998, Tiburon, CA, U.S.A.*, San Francisco, 1999. Morgan Kaufmann.
- [9] Vasken Bokossian, Chenggong C. Fan, Paul S. LeMahieu, Marc D. Riedel, Lihao Xu, and Jehoshua Bruck. Computing in the RAIN: A Reliable Array of Independent Nodes. In *IEEE Transactions on Parallel and Distributed Systems*, volume 12. IEEE, February 2001.
- [10] D. Caromel and J. Vayssiere. A Java framework for seamless sequential, multi-threaded, and distributed programming. In *ACM 1998 Workshop on Java for High-Performance Network Computing*, INRIA Sophia Antipolis, Greece, 1998.
- [11] Trishul M. Chilimbi, Mark D. Hill, and James R. Larus. Cache-Conscious Structure Layout. In *Proceedings of SIGPLAN'99 Conference on Programming Languages Design and Implementation*, ACM SIGPLAN Notices, pages 1–12, Atlanta, May 1999. ACM Press.
- [12] Jong-Deok Choi, Manish Gupta, Mauricio Serrano, Vugranam C Sreedhar, and Sam Midkiff. Escape Analysis for Java. In *ACM SIGPLAN Conference on Object Oriented Programming Systems, Languages, and Applications*. ACM Press, November 1999.
- [13] D.Manivannan and M. Singhal. Quasi-Synchronous Checkpointing: Models, Characterisation, and Classification. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):703–713, July 1999.
- [14] Zhen He, Stephen M Blackburn, Luke Kirby, and John N Zigman. Platypus: Design and Implementation of a Flexible High Performance Object Store. In *Ninth International Workshop on Persistent Object Systems: Design, Implementation and Use*, Lillehammer Norway, 2000. Springer.
- [15] Zhen He and Alonso Marquez. Cache Conscious Clustering C3. In Heinrich C. Mayr, Jirí Lazanský, Gerald Quirchmayr, and Pavel Vogel, editors, *DEXA*, volume 2113 of *Lecture Notes in Computer Science*, pages 815–825. Springer, 2001.
- [16] Zhen He, Alonso Marquez, and Stephen Blackburn. Opportunistic On-line Clustering and Statistic Collection. In D S Munro, editor, *Proceedings of the Seventh IDEA International Workshop*, Victor Harbour, South Australia, February 2000.
- [17] Zhen He, Jeffrey Xu Yu, and Stephen Blackburn. Object Placement in Shared Nothing Architecture. In Stephen M Blackburn, editor, *Proceedings of the Sixth IDEA International Workshop*, Rutherglen, Australia, January 1999.
- [18] Fabrice Le Fessant, Ian Piumarta, and Marc Shapiro. An Implementation for Complete, Asynchronous, Distributed Garbage Collection. In *Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation (PLDI)*, volume 33 of *SIGPLAN Notices*, pages 152–161, Montreal, Canada, May 1998. ACM.
- [19] M.J.M. Ma, F.C.M. Lau C.L. Wang, and Z. Xu. JESSICA: Java-Enabled Single System Image Computing Architecture. In Ronald Morrison, Mick Jordan, and Malcolm Atkinson, editors, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 99)*, Las Vegas, July 1999.

- [20] Jan-Willem Maessen, Arvind, and Xiaowei Shen. Improving the java memory model using CRF. In *Conference on Object-Oriented*, pages 1–12, 2000.
- [21] Alonso Marquez, John N Zigman, and Stephen M Blackburn. Fast Portable Orthogonally Persistent Java. *Software: Practice and Experience*, 30(4):449–479, April 2000.
- [22] Stuart Norcross, Ron Morrison, Dave Munro, and Henry Detmold. Implementing a Family of Distributed Garbage Collectors. In *Proceedings of ACSC*, 2003.
- [23] William Pugh. Fixing the java memory model. In *Java Grande*, pages 89–98, 1999.
- [24] Florin Sultan, Thu D. Nguyen, and Liviu Iftode. Lazy Garbage Collection of Recovery State for Fault-Tolerant Distributed Shared Memory. In *IEEE Transactions on Parallel and Distributed Systems*, number 10, pages 1085–1098, October 2002.
- [25] The Australian National University. The Bunyip Beowulf Project. <http://tux.anu.edu.au/Projects/Bunyip/>.
- [26] John N Zigman and Ramesh Sankaranarayana. Designing a distributed JVM on a cluster. In *Proceedings of the 17th High Performance and Large Scale Computing Conference*, Nottingham, United Kingdom, 2003. Submitted, under consideration.