Distributed Configuration Management for Reconfigurable Cluster Computing

Aju M. Jacob, Ian A. Troxel, and Alan D. George
{jacob, troxel, george}@hcs.ufl.edu

High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering, University of Florida
Gainesville, Florida 32611-6200

Abstract

Cluster computing offers many advantages as a highly cost-effective and often scalable approach for high-performance computing in general, and most recently as a basis for hardware-reconfigurable systems. To achieve the full potential of performance of reconfigurable HPC systems, a runtime configuration service is required. Centralized configuration services are a natural starting point but tend to limit performance and scalability. For large-scale RC systems, the configuration service must be optimized for the system topology and management scheme. This paper presents the CARMA framework for reconfigurable clusters including several distributed configuration management schemes that leverage high-speed networking. The experimental results from this paper highlight the affects of the design of this configuration service and provide a comprehensive performance analysis of two of the proposed management schemes.

Keywords: Reconfigurable Computing, High-Performance Computing, Cluster Computing, Run-Time Reconfiguration, Scalable Coherent Interface.

1. Introduction

A trend toward high-performance cluster computing based largely on Commercial-off-the-Shelf (COTS) technologies has recently developed within the reconfigurable computing (RC) community. The creation of a 48-node RC cluster at the Air Force Research Laboratory in Rome, NY [1] is evidence of this trend. Indeed, COTS cluster computing has the potential to provide a cost-effective, scalable platform for high-performance parallel RC. However, while the addition of RC hardware has improved the performance of many stand-alone applications, providing a versatile multiuser and multitasking environment for clusters of RC and conventional resources imposes additional challenges.

While many of these "new" challenges bear a striking resemblance to traditional high-performance computing (HPC) problems, which will likely have similar solutions, others have very little correspondence whatsoever. One such example is the task of dynamically providing numerous configurations to distributed RC resources in an efficient manner. At first glance, changing computation hardware during execution in RC systems has no traditional HPC analog. However, future development of an RC programming model to allow the reuse of configuration files through run-time libraries resembles the concept of code reuse in tools such as the International Mathematical and Statistical Library (IMSL). This collection of mathematical functions abstracts the low-level coding details from developers by providing a means to pass inputs between predefined code blocks [2]. While tools like IMSL provide this abstraction statically, future RC programming models will likely include run-time library access. Core developers are providing the basis for such libraries [3].

While the programming model aspect of configuration files could relate to traditional HPC, other aspects may not. For example, one might equate the transmission of configuration files to data staging by assuming they are simply another type of data necessary to be loaded onto a set of nodes at run-time. One reason this approach does not hold is the adaptable nature of configuration files. Recent work has suggested an adaptive algorithms approach to RC in which tasks may require the run-time use of hundreds of configuration file versions [4]. Traditional HPC systems are not designed to recompile code at run-time much less do so hundreds of times per job.

Another reason configuration files differ is that, while relatively small, the amount of high-speed cache dedicated to their storage is typically small, if existent. While some FPGAs allow up to four configurations to be stored on chip [5], systems that do not allocate on-chip or on-board memory cannot preemptively "stage" configurations at all. As demonstrated, configuration management and other issues need to be considered to achieve a versatile platform for RC-based HPC, especially if such systems may one day include grid-level computation where communication latencies can lead to significant performance penalties.

To address configuration management and many other key issues facing clustered HPC/RC designs, the HCS Research Lab at Florida proposes the Comprehensive Approach to Reconfigurable Management Architecture (CARMA) framework [6]. CARMA provides a framework to develop and integrate key components as show in Figure 1. With CARMA, the RC group at HCS seeks to specifically address key issues such as: dynamic RC fabric discovery and management; coherent multitasking in a versatile multi-user environment; robust job scheduling and management; fault tolerance and scalability; performance monitoring down into the RC fabric; and automated application mapping into a management tool. This paper

focuses on the Configuration Management (CM) portion of CARMA's *RC Cluster Management* module in order to highlight the design and development of distributed CM schemes.

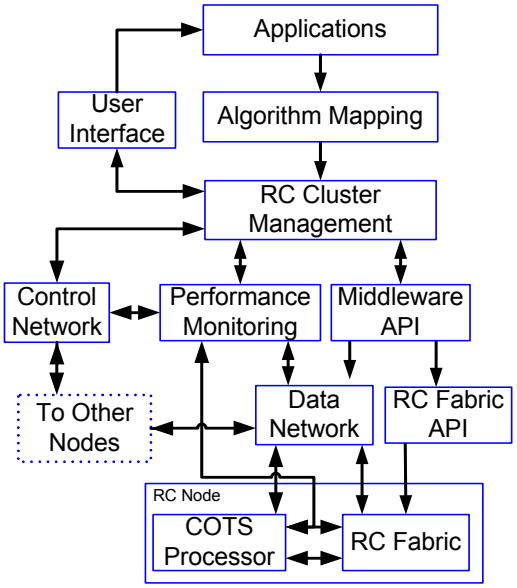


Figure 1. The CARMA Framework

The remaining sections of this paper are organized as follows. Section 2 provides a background of past work in configuration management services while Section 3 describes the design and development of CARMA's CM. Section 4 provides a detailed discussion of the experimental setup while Section 5 presents and analyses the experimental results. Finally, Section 6 describes conclusions and future work.

2. Background

A general background of RC is not included here, as such a discussion has been covered by many sources such as Compton and Hauck [7]. As described in Section 1, FPGA configuration at run-time is one of the most critical processing components that must be handled with care to ensure an RC speedup over traditional processors. Configuring RC resources is pure overhead in any RC system and thus has the potential to overshadow RC performance gains.

There have been many noteworthy projects aimed at creating a CM framework upon which CARMA's CM builds, but only two will be discussed briefly due to space limitations: RAGE from the University of Glasgow [8] and a reconfiguration manager from Imperial College, UK [9]. In addition, CARMA also expands upon the emerging FPGA O/S concept [10-13] as well as many other concepts but with a scalable, distributed, fault-tolerant focus. However, for the purposes of this paper, the background discussion will be limited to the CM.

The RAGE system provides a high-level interface for applications to perform complex reconfiguration and circuit manipulation tasks. Figure 2 shows the dataflow of the RAGE system. A Virtual Hardware Manager (VHM) orchestrates the system by accepting application descriptions, requesting circuit transforms if they do not currently fit in the FPGA, managing a circuit store and interacting with the configuration manager and device driver.

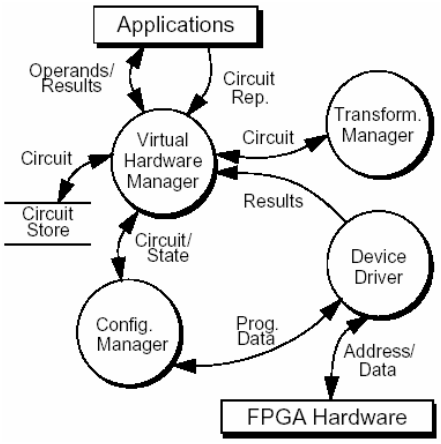


Figure 2. RAGE System Data Flow [8]

The reconfiguration manager framework from Imperial College is shown in Figure 3. This CM framework is composed of three main components: the Configuration Store, Loader, and Monitor. The Configuration Store contains the set of configuration files available to the system. The Loader, upon receiving a request from the Monitor, loads the chosen configuration onto the FPGA while the Monitor determines when the FPGA must be reconfigured.

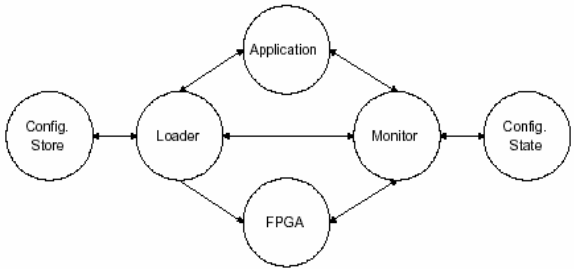


Figure 3. Imperial College Framework [9]

The CARMA framework analog to the RAGE's VHM is the Job Scheduler in the *RC Cluster Management* box in Figure 1. CARMA's version extends this philosophy by including distributed job scheduling and fault tolerance through checkpoint and rollback. CARMA's CM extends the RAGE CM by considering a distributed manager and circuit store. A more detailed description of CARMA's CM is given in Section 3. Imperial College's Monitor is analogous to CARMA's Performance Monitor. The Gossip-Enabled Monitoring Service (GEMS) [14] developed at Florida is being extended to bring robust and highly

responsive monitoring down into the FPGAs resources for CARMA.

3. Configuration Manager Design

A functional description of how the CM interacts with other CARMA concepts is given in Figure 4. The CM receives configuration requests from the *Job Scheduler*. Upon receiving a request, the *File Location* module attempts to locate the configuration file in the node's local cache. If there is a miss, a query is sent to a remote CM of a master, server, broker or peer depending on the management scheme used. A more detailed description of these distributed management schemes is given in Section 3.1. The *File Transport* layer packages the configuration file and transfers it over the data network. TCP/IP and Scalable Coherent Interface (SCI) [15] are currently supported with collective communication mechanisms such as multicast [16]. The *File Managing* layer is responsible for managing FPGA resource access and defragmentation [17], as well as configuration caching [18], relocation and transformation [19]. The *File Loading* layer uses a library of board-specific functions to configure and control the RC board(s) in the system and provide hardware independence to higher layers.

Figure 5 shows how CMs are interconnected between nodes. A communication module handles all inter-node communication between CMs and all other CARMA modules. TCP sockets (e.g. over IP over Gigabit Ethernet) comprises the control network, while SCI currently serves as the data network for configuration file transfers. Modules within a node use a form of inter-process communication (i.e. message queues) to pass requests and status. All modules have been developed as separate processes, rather than inter-related threads, in order to increase the fault tolerance of the system.

3.1. Distributed CM Schemes

In order to provide a scalable, fault-tolerant CM service for thousands of nodes (one day) the CARMA CM is fully distributed. In creating the distributed CM model, four distributed management schemes are proposed: *Master-Worker* (MW), *Client Server* (CS), *Client Broker* (CB), and *Peer-to-Peer* (PP). Figure 6 illustrates these four schemes. While CM modules exist in different forms on various nodes in the four schemes, in all cases the CMs still communicate with one another as defined above.

The MW scheme (Figure 6a) is a centralized scheme where the master maintains a global view of the system and has full control over job scheduling and configuration management. This scheme is representative of currently proposed CMs. While a centralized scheme is easy to implement, there will be performance limitations due to poor scalability for systems with a large number of nodes. The other three schemes in Figure 6 assume a distributed job scheduling service. For the CS scheme, (Figure 6b) local CMs request and receive configurations from a server. Although this scheme is likely to exhibit better performance than MW for a given number of nodes, there will also be scalability limitations as the number of nodes is increased.

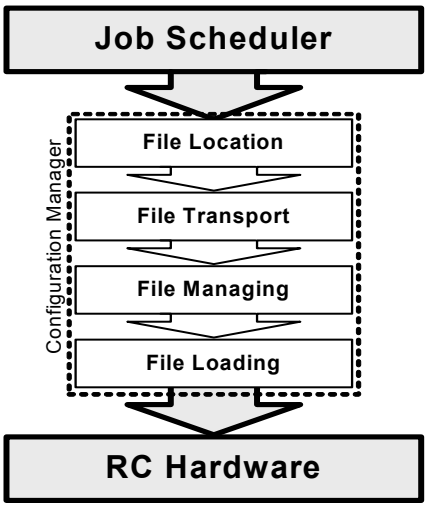


Figure 4. CM Layered Design

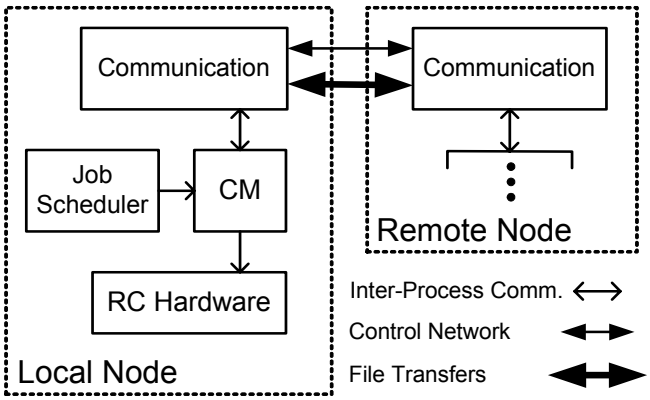


Figure 5. CM Inter-node Communication

The CB scheme (Figure 6c) is a low-overhead version of CS because Local CMs receive a pointer from the broker to the client node that possesses the requested configuration. This scheme will likely further reduce the server bottleneck by reducing service time. Having multiple servers/brokers may further reduce these limitations. The PP scheme (Figure 6d) contains fully distributed CMs where there is no central view of the system. This scheme will likely provide better performance when the number of nodes is rather large (in order to overcome the added complexity). A hierarchical-layered combination of these four schemes will likely be needed to provide scalability up to thousands of nodes. For example, nodes are first grouped using CS or CB and then these groups are grouped using PP. The layered group concept has been found to dramatically improve scalability of HPC services for thousands of nodes [14]. The following section will detail the experimental setup used to evaluate and compare the MW and CS schemes; CB and PP are reserved for future study.

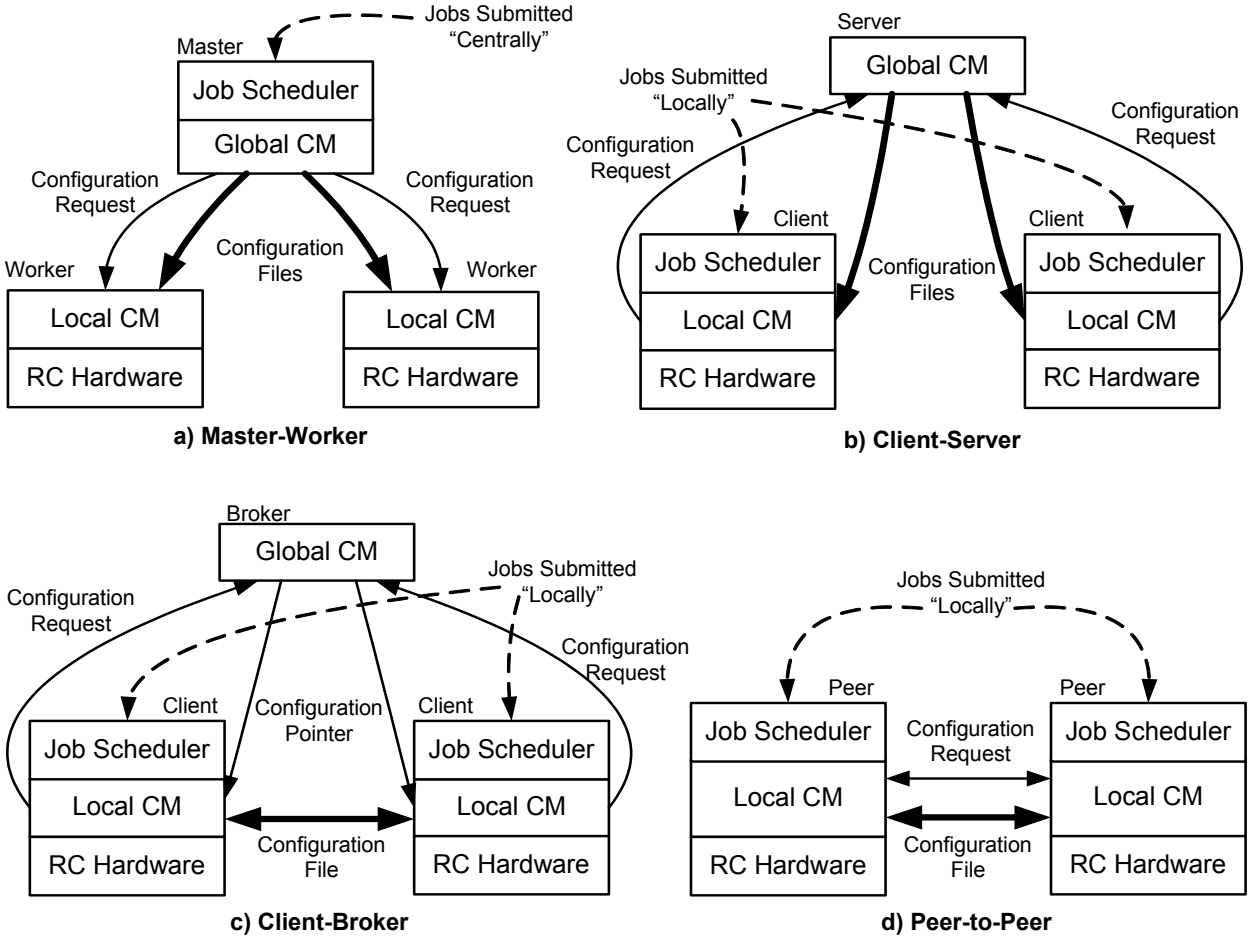
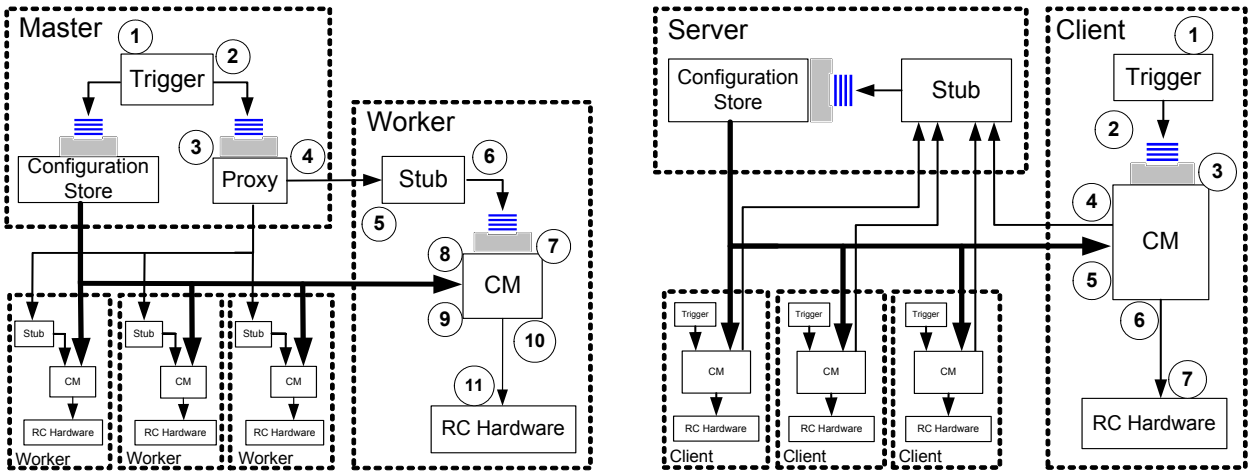


Figure 6. Distributed Configuration Management Schemes

4. Experimental Setup

To investigate the performance of the MW and CS schemes, a series of experiments have been conducted. The objectives of these performance experiments are to determine the overhead imposed on the system by the CARMA CM, determine the components of latency in configuration transactions, and to provide a quantitative comparison between MW and CS. The performance metric used to compare these two schemes is defined as the completion latency from the time a configuration request is received from the job scheduler until the time that configuration is loaded onto the FPGA.

Figures 7a and 7b illustrate the individual actions that compose a configuration transaction indicating the points at which latency has been measured for MW and CS, respectively. The experimental setup for each scheme has one master or server in the system and four worker or client nodes. The *Trigger* block in both diagrams of Figure 7 acts in place of the CARMA *Job Scheduler* in order to stimulate the system in a simple periodic manner. A MW configuration transaction is composed of the following components as shown in Figure 7a. The interval from 1 to 2 is defined as *Creation Time* and is the time it takes to create a configuration request data structure. The interval from 2 to 3 is defined as *Proxy Queue Time* and is the time the request waits in the proxy queue until it can be sent over a TCP connection to the worker, while the *Proxy Processing Time* is the time it takes the Proxy to create a TCP connection to the worker and is the interval from 3 to 4. These connections are established and destroyed with each transaction because maintaining numerous connections is not scalable. The interval from 4 to 5 is defined as *Request Transfer Time* and is the time it takes to send a configuration request of 148 bytes over TCP. This delay was observed to average 420µs using TCP/IP over Gigabit Ethernet with a variance of less than 1%. The interval from 5 to 6 is defined as *Stub Processing Time* and is the time it takes the Stub to read the TCP socket and place it in the CM queue, while the interval from 6 to 7 is defined as the *CM Queue Time* is the time the request waits in the CM queue until the CM removes it. The *CM Processing Time* is the time



a) Master-Worker **b) Client-Server**

Figure 7. Configuration Request Flow for MW and CS Schemes

required to accept the configuration request and determine how to obtain the needed configuration file and is the interval from 7 to 8. The interval from 8 to 9 is defined as *File Retrieval Time*, the time it takes to acquire the configuration file over SCI, including connection setup and tear down, whereas the interval from 9 to 10 is defined as *CM-HW Processing Time* and is the time to set up the board-specific API with the configuration file. Finally, the interval from 10 to 11 is defined as *HW Configuration Time* and is the time it takes to configure the FPGA.

CS configuration transactions are composed of the following components as shown in Figure 7b. Note that jobs (via the *Trigger* block) are created on the client nodes in the CS scheme rather than on the master in the centralized MW scheme. The intervals from 1 to 2, 2 to 3 and 3 to 4 are the *Creation Time, CM Queue Time* and *CM Processing Time*, respectively and are defined as in MW. The interval from 4 to 5 is defined as *File Retrieval Time*, which is the time required for a client to send a request to the server and receive the file in response. The intervals from 5 to 6 and 6 to 7 are the *CM-HW Processing Time* and *HW Configuration Time*, respectively and are defined as in MW.

For this study, the CM was executed on five nodes, one serving as master/server and the other four as computing nodes. Each node is a Linux server with dual 2.4GHz Xeon processors and 1GB of DDR RAM. The control network between nodes is switched Gigabit Ethernet. The data network is 5.3 Gb/s SCI connected in a 2D torus (Figure 8) with Dolphin D337 cards using the SISCI software release 2.1. Each computing node contains a Tarari HPC board (i.e. CPX2100) [20] housed in a 66MHz, 64-bit PCI bus slot.

The interval between requests is chosen as the independent variable and is varied from 1s to 80ms. The value of 80ms was selected as the lower bound because this value was determined to be the minimum delay time to retrieve a Virtex-II 1000 configuration file (20ms) and configure a Tarari board (60ms). In each trial, 20 total requests were submitted with constant periodicity. The timestamps for measuring all intervals is taken using the C function *gettimeofday()* which provides a 1μs resolution.

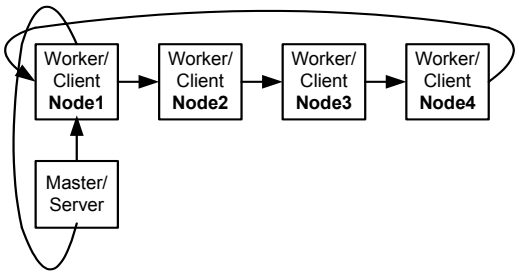


Figure 8. SCI Network Setup

5. Experimental Results

Measurements were collected and averaged across all four computing nodes for each of the components contributing to completion latency in the MW and CS schemes and are summarized in Tables 1 and 2, respectively. The results for request intervals larger than 250ms are not shown to save space. Values in this region were found to be relatively flat because the master/server is able to service requests without imposing additional delays. In this stable region, requests are only delayed by fixed values of *File Retrieval Time* and *HW Configuration Time*.

Several observations can be made from the values in Tables 1 and 2. In both MW and CS for most cases, the major components of completion latency are *CM Queue Time*, *File Retrieval Time* and *HW Configuration Time*, the largest of these naturally being *CM Queue Time* as resources become overloaded. The remaining components are in the worst case less than 1% of the total completion latency. The data demonstrates that the CM design in CARMA imposes very little overhead on the system, whereas *CM Queue Time* and *File Retrieval Time* are the most dominant components

Table 1. Average Completion Latency for Workers in MW (all values in milliseconds)

	Configuration Request Interval										
	250	200	175	150	140	130	120	110	100	90	80
Creation Time	0.016	0.014	0.013	0.012	0.012	0.013	0.013	0.014	0.013	0.014	0.013
Proxy Queue Time	0.009	0.011	0.015	0.018	0.019	0.018	0.018	0.018	0.019	0.019	0.019
Proxy Processing Time	0.014	0.013	0.011	0.010	0.009	0.009	0.008	0.008	0.007	0.007	0.007
Request Transfer Time	0.420	0.420	0.420	0.420	0.420	0.420	0.420	0.420	0.420	0.420	0.420
Stub Processing Time	0.012	0.012	0.011	0.011	0.011	0.010	0.010	0.010	0.010	0.010	0.010
CM Queue Time	*0.019*	*195*	*467*	*904*	*1155*	*1285*	*1329*	*1364*	*1590*	*1685*	*1932*
CM Processing Time	0.023	0.023	0.022	0.022	0.021	0.018	0.015	0.015	0.019	0.026	0.024
File Retrieval Time	*72*	*151*	*269*	*360*	*400*	*384*	*388*	*379*	*416*	*425*	*454*
CM-HW Processing Time	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
HW Configuration Time	60	60	60	60	60	60	60	60	60	60	60
Total Completion Latency	**132**	**406**	**797**	**1324**	**1615**	**1729**	**1777**	**1803**	**2066**	**2170**	**2445**

Table 2. Average Completion Latency for Clients in CS (all values in milliseconds)

	Configuration Request Interval										
	250	200	175	150	140	130	120	110	100	90	80
Creation Time	0.011	0.012	0.011	0.011	0.011	0.011	0.011	0.011	0.011	0.011	0.011
CM Queue Time	*0.008*	*0.007*	*0.007*	*72*	*220*	*355*	*469*	*559*	*593*	*595*	*607*
CM Processing Time	0.010	0.011	0.011	0.017	0.020	0.022	0.018	0.018	0.025	0.027	0.029
File Retrieval Time	*84*	*82*	*84*	*106*	*138*	*172*	*188*	*201*	*204*	*208*	*199*
CM-HW Processing Time	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
HW Configuration Time	60	60	60	60	60	60	60	60	60	60	60
Total Completion Latency	**144**	**142**	**144**	**239**	**418**	**586**	**716**	**821**	**856**	**863**	**866**

of completion latency when the request interval time is small. Another useful observation is the total completion latency for CS is lower than that of MW for all configuration request intervals at and below 200ms.

Figure 9 charts the completion latency values for each of the four nodes along with the average. As shown, both MW and CS schemes provide constant completion latency for configuration request intervals larger than 480ms and 200ms, respectively. A zoomed-in view of the region of interest shows MW and CS differ significantly. In all cases, CS achieves better average and per-node completion latencies than MW. However, in CS, each node experiences greater variability in completion latency due to server access. Nodes closer to the server on the SCI torus experience smaller file transfer latencies, which allows them to receive and therefore request the next configuration file faster. Completion latencies in the MW scheme have less variability between nodes because all requests are serviced and serialized in the master. Therefore, no one node receives an advantage based on proximity to the master/server.

6. Conclusions

The emergence of RC cluster computing systems will present significant challenges in resource management. While some challenges may be addressed with traditional HPC solutions, those dealing with configuration management in particular require novel approaches. The CARMA framework is being developed to help address the challenges of HPC/RC cluster management. The modular and distributed design of CARMA and its CM, and the manner in which components interact, have been described. Four distributed CM management schemes have been identified: MW, CS, CB and PP.

This study presents performance results for the MW and CS schemes in a head-to-head comparison. The results show that CARMA configuration manager imposes very little overhead on the system (i.e. less than 1% of completion latency in the worst case). The CS scheme outperforms MW in the region of interest, even with only four computing nodes. A significant variability between completion latencies of client nodes in the CS scheme is observed due to data-network proximity between the server and clients, a condition not experienced between the MW nodes. This distinction arises because in MW the configurations emerge from the central master in the order requests arrive (i.e. workers do not compete for server access) and so proximity has no affect. A centralized scheme like MW becomes more quickly saturated as workload increases and thus lacks the scalability of the CS approach.

Directions for future work include expanding and developing components within the CARMA framework. Future work for CM in particular is expected to include: extension of CS to support multiple servers; experimental analysis of the CB and PP schemes; experimentation with larger system sizes and assorted HPC/RC application studies; and analysis of torus-based versus switched-based data networks for high-speed configuration file transfers.

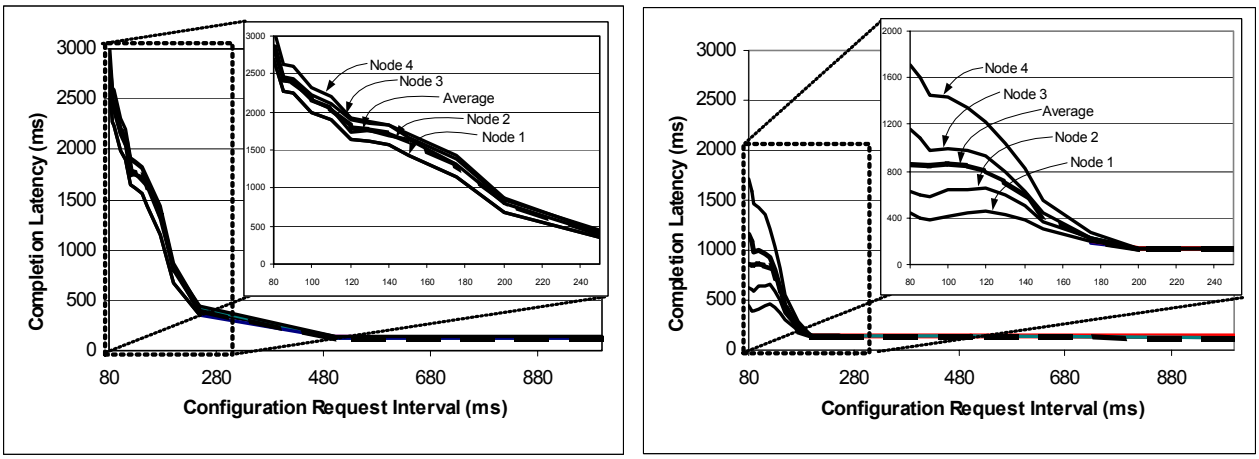


Figure 9. Average Completion Latency vs. Configuration Request Interval

7. Acknowledgments

This work was sponsored in part by the U.S. Dept. of Defense. We wish to thank the following vendors for key resources that made this research possible: Alpha Data, Celoxica, Cisco Systems, Dolphin Inter., Intel, and Tarari.

8. References

1. V. Ross, "Heterogeneous HPC Computing," *Proc. 35th GOvernment Microcircuits Applications and Critical TECHnologies (GOMACTech)*, Tampa, FL, 2003.
2. Visual Numerics, Inc., "IMSL Mathematical & Statistical Libraries", [online] 2004, http://www.vni.com/products/imsl/ (Accessed: 12 February 2004).
3. Xilinx, Inc., "Common License Consortium for Intellectual Property," [online] 2004, http://www.xilinx.com/ipcenter/ (Accessed: 12 February 2004).
4. A. Derbyshire, W. Luk, "Compiling Run-Time Parameterisable Designs," *Proc. 1st IEEE International Conference on Field-Programmable Technology (FPT)*, Hong Kong, 2002.
5. Chameleon Systems, Inc.: *CS2000 Reconfigurable Communications Processor*, Family Product Brief, 2000.
6. I. Troxel, A. George, "UF-HCS RC Group Q3 Progress Report," [online] 2004, http://www.hcs.ufl.edu/prj/rcgroup/teamHome.php (Accessed: 12 February 2004).
7. K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, Vol. 34, No.2, June 2002, pp. 171-210.
8. J. Burns, A. Donlin, J. Hogg, S. Singh, M. de Witt, "A Dynamic Reconfiguration Run-Time System," *Proc. 5th Annual IEEE Symposium on Custom Computing Machines (FCCM)*, Los Alamitos, CA, 1997.
9. N. Shiraz, W. Luk, P. Cheung, "Run-Time Management of Dynamically Reconfigurable Designs," *Field-Programmable Logic and Applications*, R.W. Hartenstein and A. Keevallik (editors), LNCS 1482, Springer, 1998, pp. 59-68.
10. R. Hartenstein, R. Kress, U. Nageldinger, "An Operating System for Custom Computing Machines based on the Xputer Paradigm," *Proc. 7th International Workshop on Field Programmable Logic (FPL)*, London, UK, 1997.
11. G. Brebner, "A Virtual Hardware Operating System for the Xilinx XC6200," *Proc. 6th International Workshop on Field Programmable Logic (FPL)*, Darmstadt, Germany, 1996.
12. O. Diessel, D. Kearney, G. Wigley, "A Web-Based Multiuser Operating System for Reconfigurable Computing," *Proc. 13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP)*, San Juan, Puerto Rico, 1999.
13. G. Wigley, D. Kearney, "The First Real Operating System for Reconfigurable Computers," *Proc. 6th Australasian Conference on Computer Systems & Architecture*, Queensland, Australia, 2001.
14. R. Subramaniyan, P. Raman, A. George, M. Radlinski, "GEMS: Gossip-Enabled Monitoring Service for Scalable Heterogeneous Distributed Systems," *White Paper*, currently in journal review, [online] 2003, http://www.hcs.ufl.edu/pubs/GEMS2003.pdf
15. D. Gustavson, Q. Li, "The Scalable Coherent Interface (SCI)," *IEEE Communications*, Vol. 34, No. 8, August 1996, pp. 52-63.
16. S. Oral, A. George, "Multicast Performance Analysis for High-Speed Torus Networks," *Proc. of 27th IEEE Conference on Local Computer Networks (LCN) via the High-Speed Local Networks (HSLN) Workshop*, Tampa, FL, 2002.
17. K. Compton, J. Cooley, S. Knol, S. Hauck, "Configuration Relocation and Defragmentation for FPGAs", *Proc. 8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa Valley, CA, 2000.
18. Z. Li, K. Compton, S. Hauck, "Configuration Caching for FPGAs," *Proc. 8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa Valley, CA, 2000.
19. Z. Li, S. Hauck, "Configuration Prefetching Techniques for Partial Reconfigurable Coprocessor with Relocation and Defragmentation," *Proc. 10th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, Monterey, CA, 2002.
20. Tarari Inc., "High-Performance Computing Processors Product Brief," [online] 2004, http://www.tarari.com/PDF/HPC-BP.pdf (Accessed: 12 February 2004).