

Distributed Components for Integrating Large-scale HPC Applications

Nanbor Wang and Johan Carlsson

{nanbor, johan}@txcorp.com

Tech-X Corporation

5621 Arapahoe Avenue, Suite A

Boulder, CO 80303

Keywords: Component-based software development, common component architecture, high performance computing, framework, parallel and distributed computing.

1 Introduction

Component-based software engineering (CBSE) [1, 2] is now a widely accepted software paradigm for developing large, complex enterprise applications. CBSE promotes software reuse and simplifies building and maintaining these large applications by separating different aspects of software development, *e.g.* component interaction models, component implementations, and component composition, into individual entities that can easily be maintained [3]. There exist many popular enterprise component standards and implementations for different application domains. Examples include Java Beans [4] and ActiveX [5] for building single executables, and CORBA Component Model (CCM) [6], Enterprise Javabeans (EJB) [7] and Distributed Component Object Model (DCOM) [8] for building large-scale distributed applications.

Existing component standards and frameworks, however, are designed with enterprise applications in mind and do not address the needs for High-Performance Computing scientific applications such as combustion modeling, global climate modeling, and fusion plasma simulation. As a result, these existing frameworks do not support features that are important for HPC scientific applications, such as the interoperability with scientific programming languages (FORTRAN) and parallel computing infrastructure (message passing standards, such as MPI). The DOE-sponsored Common Component Architecture (CCA) [9] was developed as a grass-roots effort to address these needs of the HPC communities before they may be adequately served by the commodity off-the-shelf software frameworks and tools [10].

In this paper, we demonstrate the needs for distributed

components in building large-scale, distributed and parallel HPC applications. We also present our ongoing work in developing distributed CCA components. Our work is built on top of Babel/Ccaffeine parallel CCA tools and existing promoting technologies. We then review related work. Finally, we present our concluding remarks and sketch the future direction for our research.

2 The Case for Distributed HPC Components

Many existing CCA implementations, such as Ccaffeine [11, 12] and Dune [13], are research frameworks aiming to bring CBSE into the high-performance parallel computing world by composing local components developed with different languages in the same address space. The basic language interoperability needs (specifically, modern FORTRAN languages) are often handled by tools like the Babel [14] Scientific Interface Description Language (SIDL) compiler and Chasm [15] Language Interoperability Tools. These frameworks, however, do not currently support distributed components.

Large-scale scientific computing projects, such as the Fusion Simulation Project (FSP), can benefit from connecting and utilizing remote computing resources. The following use cases exemplify possible use cases of integrated distributed and parallel applications.

- Due to restrictions on resources at supercomputing centers, it is not uncommon for users to perform only the computationally intensive work at those machines and transport the resulting data back to local clusters or desktop machines for post processing, data analysis, and visualization. Resource restrictions can also be caused by a lack of software tools on supercomputers commonly used by scientists. Likewise, data visualization is often not done remotely at a supercomputer

due to a lack of hardware supports at supercomputing center and the inefficiency incurred by running remote visualization over the X protocol. Connecting numerical computation jobs with remote data analysis and visualization applications allows users to streamline the workflow of scientific explorations.

- Another approach commonly adopted by computational scientists is to handle the data generation, analysis, and visualization in batch mode. Using this approach, users need to *buffer* the generated data files locally in the storage available at the supercomputing facility. However, most users often have a limited disk quota for storing generated data. As a result, users are forced to decide what data is interesting for later analysis and discard the remaining data before running the next computational job. To avoid wasting precious computational cycles, it is sometimes suitable to collect as much data as possible to explore new scientific discoveries by streaming all the interesting data out of the supercomputer to a remote data storage *in real time* [16].
- Parallel physics components are not necessarily optimized for the same parallel architecture. For example, one physics component might be internally tightly coupled and require a shared-memory machine to run well. Another physics components might be able to take advantage of multi-level parallelism and run best on a parallel vector machine. A third physics component might not benefit at all by running on anything but a standard Beowulf cluster. In cases like these, a loosely coupled integrated physics application composed using distributed components would allow optimal use of computational resources.

3 Exploring Diverse Distributed Technologies

Figure 1 presents an achitectural overview of how distributed components can be used to build distributed applications. Distributed components are CCA components that utilize exiting distributed technologies under the hood to bring distributed aspects to conventional HPC applications. Several existing CCA framework implementations, such as XCAT [17] and SciRUN [18], do provide distributed component capability. However, they do not currently interoperate with each other or other CCA frameworks that target running high-performance parallel components. For example, the XCAT project focuses on providing a distributed com-

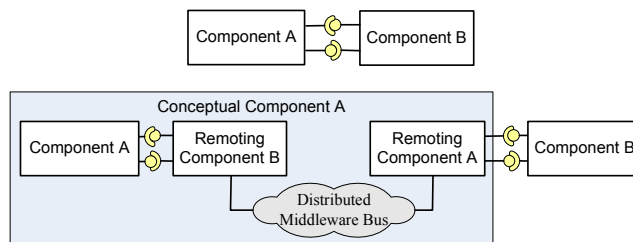


Figure 1: **Supporting Distributed Application with Distributed Components.**

ponent framework and uses Web Services as its underlying distributed technology. SciRUN, however, uses CORBA as its distributed backbone.

To support the kind of distributed and parallel component-based applications outlined in the previous section, we are working on implementing distributed CCA components using a variety of existing distributed technologies which cover the distributed connecting aspects that existing HPC CCA frameworks do not address. This approach will accommodate all the existing CCA implementations and to provide connecting mechanisms for existing CCA frameworks to talk to each other. To implement such distributed components, the two connecting components will implement the matching port for the target components to which the distributed component is connecting. Existing distributed middleware standards can then be adopted to provide the connectivity between distributed components. This approach utilizes the easy composability of component framework and allow us to connect applications of the same or different CCA implementations.

For example, a developer can stream the data generated by a simulation job running on a supercomputer to an in-house cluster for data analysis or visualization using distributed components. Multiple distributed component pairs can be used along with MxN components to establish multiple data streams between the supercomputer and the local cluster. These multiple data streams effectively expand the TCP window size and allow scientist to take advantage of high-bandwidth, high-latency networks that are increasingly common among major research institutes [19].

Since our approach utilizes existing and mature distributed technologies, implementing distributed components also enables scientists access to abundant, existing distributed services available on the network. For example, a Fusion Data Grid Service is being developed to allow fusion plasma scientists access to experimental and simulation data via a set of unified user interfaces. This service will enable scientists easy access to these data and to reconcile exper-

Table 1: Comparing Local Functions Calls with Ccaffeine Calls

Type	Direct Calls	Virtual Calls	CCA Calls
calls/sec	103M	82.6M	36.5M

imental and simulation data. We will be able to make services like this available to CCA applications as a distributed component.

We have implemented a prototype distributed component using widely accepted distributed middleware, such as CORBA [20] and Web Services. To help us better evaluate and select the proper distributed technologies, we are conducting a series of benchmarking experiments under different usage scenarios. We have thus far measured a basic component interaction model. A function `double cube_double (in double x)`, which returns the cube of input number `x`, is used to show how fast components can exchange frequent and short messages.

The benchmark experiments use Ccaffeine 0.5.6 as the CCA framework on a single 2.8 GHz Intel Pentium-4 CPU running Red Hat Linux with kernel version 2.4.20-8. All components are compiled with gcc 3.4.2 compilers using full optimization (`-O3`). Table 1 summarizes our first set of measurements comparing the baseline performance of calling `cube_double` between local Ccaffeine components to simple, non-inline C++ function calls and virtual function calls. This experiment magnifies the cost of function calls by reducing the time spent in the actual function. Although these experiments have been performed previously by other CCA researchers, we believe it is important to conduct the same tests again to keep track of the performance of current Babel/Ccaffeine implementations and to establish a basic performance index of the machines used for subsequent experiments.

We have also implemented a set of distributed components using TAO [21], a high performance CORBA implementation developed by Washington University in St. Louis and Vanderbilt University and gSoap, a popular C-based Web Services development framework by Dr. Robert A. van Engelen of Florida State University. To magnify the time spent on processing remote invocation, the experiments were conducted on the same machine, over the local loop-back interface. This reduces the time spent on shipping data over the network and thus signifies the time for marshaling and demarshaling data for remote invocations.

As we can see from Table 2 and Table 1, implementing distributed components as Babel/Ccaffeine components imposes only minimal overhead to the component interac-

Table 2: Comparing the Performance of Operation Calls Between Simple Client/Server Pair and Distributed Components Using Different Distributed Technologies (in calls/sec)

Type	Client/Server	Distributed Component
CORBA (TAO)	8674	8386
Web Services (gSOAP)	6162	6030

tion. For coarser-grained components that often perform more complex computations, these overheads are negligible. Based on our previous work on sending large datasets over the network using CORBA and gSOAP [22], we know that when sending large amount of data over the wire, the overhead incurred by the HPC CCA framework will be even more insignificant. We do plan to conduct more investigation into the issue of exchanging distributed large datasets with our distributed component approach.

4 Related Work

There are several ongoing efforts in the CCA community to support distributed CCA applications. As we already mentioned, the XCAT project developed by the Indiana University concentrates on exploring a CCA infrastructure over a distributed model. It currently does not interact with other CCA implementations.

There is also ongoing effort in the Babel group to support Remote Method Invocation [23] for parallel data redistribution. The effort does not specifically depend on any specific distributed middleware technologies. Therefore, our work will be complementary to their effort in selecting a proper distributed technology under different usage scenarios. Unlike Babel RMI work, where all interactions are modeled between components, distributed components allow scientists to abstract existing distributed services not implemented in CCA as local CCA components.

The SciRUN team is developing the meta-component model in SciRUN-2 project. It will provide interoperability between Ccaffeine/Babel components inside a SciRUN-2 runtime. Like our distributed components, this interoperability will enable scientists to take advantage of Babel RMI work and connect numeric computation with remote SciRUN data processing and analysis process. The Portable Parallel CORBA Object (Paco++) [24] in the Paris Project developed by la Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), France, takes a different ap-

proach to facilitate deploying parallel objects over a distributed network.

5 Conclusions and Future Work

The DOE sponsored CCA efforts address the lack of support for the HPC community from commercial component based frameworks. Existing CCA implementations, however, do not interoperate well with each other. The distributed components that we are developing enable scientists to integrate and connect components from different CCA frameworks, to utilize distributed network resources for data storage or post processing, to connect to existing distributed services, and to compose loosely coupled applications where each component is running on its optimal parallel architecture.

We have implemented a prototype distributed component using CORBA and performed preliminary benchmarking. We plan to continue the prototyping and benchmarking effort in the near future to explore other distributed technologies, such as Web Services, and their performance under other interaction scenarios, such as exchange of large datasets. These prototyping and benchmarking efforts will be beneficial for both our subsequent research and to the broader CCA community.

In our Phase II research, our focus will shift toward applying the distributed components to provide transparent connections between remote high performance applications, e.g. connecting supercomputers running numerical computations with local clusters running data analysis and visualization, or connecting physics components running across different types of parallel machines.

Acknowledgements

This research is funded by DOE Office of Advanced Scientific Computing Research (grant # DE-FG02-04ER84099) and by Tech-X Corporation. We would like to express our thanks to the CCA community for support and fruitful discussions.

References

- [1] G. Heineman and B. Councill, eds., *Component-Based Software Engineering*. Reading, Massachusetts: Addison-Wesley, 2000.
- [2] C. Szyperski, *Component Software—Beyond Object-Oriented Programming*. Santa Fe, NM: Addison-Wesley, 1998.
- [3] N. Wang, D. C. Schmidt, A. Gokhale, C. Rodrigues, B. Natarajan, J. P. Loyall, R. E. Schantz, and C. D. Gill, “QoS-enabled Middleware,” in *Middleware for Communications* (Q. Mahmoud, ed.), New York: Wiley and Sons, 2004.

- [4] “Java beans.” <http://java.sun.com/products/javabeans/>.
- [5] C. Egremont, III, *Mr. Bunny’s Guide to ActiveX*. Reading, MA: Addison-Wesley, 1998.
- [6] Object Management Group, *CORBA Components*, OMG Document formal/2002-06-65 ed., June 2002.
- [7] Sun Microsystems, “Enterprise JavaBeans Specification.” java.sun.com/products/ejb/docs.html, Aug. 2001.
- [8] Microsoft Corporation, *Distributed Component Object Model Protocol (DCOM)*, 1.0 ed., Jan. 1998.
- [9] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski, “Toward a Common Component Architecture for High-Performance Scientific Computing,” in *Proceedings of IEEE Symposium on High Performance Distributed Computing*, (Redondo Beach, CA), IEEE, 1999.
- [10] D. E. Bernholdt, B. A. Allan, R. Armstrong, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumpfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. McInnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, and S. Zhou, “A component architecture for high-performance scientific computing,” *Intl. J. High-Perf. Computing Appl.*, 2005. Submitted to ACTS Collection special issue.
- [11] B. Allan, R. Armstrong, S. Lefantzi, J. Ray, E. Walsh, and P. Wolfe, “Ccaffeine – a CCA component framework for parallel computing.” <http://www.cca-forum.org/ccafe/>.
- [12] B. A. Allan, R. C. Armstrong, A. P. Wolfe, J. Ray, D. E. Bernholdt, and J. A. Kohl, “The CCA core specification in a distributed memory SPMD framework,” *Concurrency and Computation: Practice and Experience*, vol. 14, no. 5, pp. 323–345, 2002.
- [13] C. D. Rickett, S.-E. Choi, C. E. Rasmussen, and M. J. Sottile, ““rapid prototyping frameworks for developing scientific applications: A case study”,” in *Proceedings of the Los Alamos Computer Science Institute (LACSI) Symposium*, (Santa Fe, NM), oct 2004.
- [14] T. Dahlgren, T. Epperly, G. Kumpfert, and J. Leek, *Babel User’s Guide*. CASC, Lawrence Livermore National Laboratory, Livermore, CA, babel-0.9.4 ed., 2004.
- [15] A. C. L. (LANL), “Chasm – language interoperability tools.” <http://chasm-interop.sourceforge.net/>.
- [16] V. Bhat, S. Klasky, S. Atchley, M. Beck, D. McCune, and M. Parashar, “High performance threaded data streaming for large scale simulations.” in *GRID*, pp. 243–250, 2004.
- [17] <http://www.extreme.indiana.edu/xcat/>.
- [18] <http://www.sci.utah.edu/>.
- [19] “Energy science network.” <http://www.es.net/>.
- [20] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*. Reading, MA: Addison-Wesley, 1999.
- [21] I. Pyarali, C. O’Ryan, D. C. Schmidt, N. Wang, V. Kachroo, and A. Gokhale, “Using Principle Patterns to Optimize Real-time ORBs,” *IEEE Concurrency Magazine*, vol. 8, no. 1, 2000.
- [22] S. G. Shasharina, N. Wang, and J. R. Cary, “Grid service for visualization and analysis of remote fusion data,” in *Proceedings of Challenges of Large Applications in Distributed Environments (CLADE)*, (Honolulu, HI), jun 2004.
- [23] “Parallel data redistribution.” <http://www.llnl.gov/CASC/components/parallel.html>.
- [24] Sébastien Lacour and Christian Pérez, “Automatic deployment of MPI applications on a computational grid,” in *Proceedings of Computational Science and Engineering of the Society for Industrial and Applied Mathematics (SIAM-CSE2005)*, (Orlando, FL), Feb. 2005.