CCALoop: Scalable Design of a Distributed Component Framework

Kostadin Damevski, Ashwin Deepak Swaminathan, and Steven Parker
Scientific Computing and Imaging Institute
University of Utah, Salt Lake City, Utah 84112, USA
{damevski,ashwinds,sparker}@sci.utah.edu

Categories and Subject Descriptors

D.4.7 [**Software Operating Systems**]: Organization and Design—*Distributed systems*

General Terms

Design, Management, Reliability

1. INTRODUCTION

In recent years, component technology has been a successful methodology for large-scale commercial software development. Component technology encapsulates a set of frequently used functions into a component and makes the implementation transparent to the users. Application developers typically use a group of components, connecting them to create an executable application. The components are managed by a component framework that exists on each computing node where components may be instantiated or executed. A component framework provides a set of services to components: locating other components in the system, instantiating, connecting, executing, reporting error messages or results, etc. It can also provide a user interface, often a Graphical User Interface (GUI), to compose, execute and monitor components. In order to manage a large component application that uses many components and utilizes sets of distributed computing resources, one or more component frameworks have to exist on each separate computing resource. This requires that multiple frameworks cooperate in some fashion to manage and monitor a large component application.

Component technology is becoming increasingly popular for large-scale scientific computing in helping to tame the software complexity required in coupling multiple disciplines, multiple scales, and/or multiple physical phenomena. The Common Component Architecture (CCA) [2] is a component model that was designed to fit the needs of the scientific computing community by imposing low overhead and supporting parallel components. The CCA standard also provides for the inclusion of Single Program Multiple Data (SPMD) parallel components. These components exists in several address spaces and are internally managed by message passing (e.g. Message Passing Interface (MPI)). A compliant framework needs to provide the facilities to instantiate, manage, and execute this novel type of component. The CCA model selects a lightweight component framework that optimizes execution efficiency. Several software frameworks are targeted at the CCA component model, including SCIRun2 [7], CCAFFEINE [1], XCAT [4] and others. Scientific computing frameworks not based on CCA are also popular and have relatively similar structure. All of these systems enable creation of complex high-performance simulations through the assembly of software components. Component frameworks aimed at scientific computing need to support a growing trend in this domain toward larger simulations that produce more encompassing and accurate results. The CCA component model has already been used in several domains, creating components for large simulations involving accelerator design, climate modeling, combustion, and accidental fires and explosions [5]. These simulations are often targeted to execute on sets of distributed memory machines spanning several computational and organizational domains. To address this computational paradigm a collection of component frameworks that are able to cooperate to manage a large, long-running scientific simulation containing many components are necessary.

A component framework's data is queried and modified by a user (through a GUI) and by executing components. In both cases it is imperative that the framework provides quick responses under heavy loads and high-availability to long running applications. The goal of our work is to present a solution to several key issues is distributed component framework design. The system described in this paper is architected to: (1) scale to a large number of nodes and components, (2) maintain framework availability when framework nodes are joining and leaving the system and be able to handle complete node failures, (3) facilitate multiple human users of the framework, (4) support the execution and instantiation of SPMD parallel components. Our distributed component framework, CCALoop, is self-organizing and uses an approach that partitions the load of managing the components to all of the participating distributed frameworks. The responsibility for managing framework data is divided among framework nodes by using a technique called Distributed Hash Tables (DHT) [3]. CCALoop uses a hash function available at each framework node that maps a specific component type to a framework node in a randomly distributed fashion. This operation of mapping each component to a node is equally available at all nodes in the system. Framework queries or commands require only one-hop routing in CCALoop. To provide one-hop lookup of framework data we keep perfect information about other nodes in the system, all the while allowing a moderate node

HPDC'07, June 25–29, 2007, Monterey, California, USA.
ACM 978-1-59593-673-8/07/0006.

joining/leaving schedule and not impacting scalability. We accommodate the possibility that a framework node may fail or otherwise leave the system by creating redundant information and replicating this information onto other frameworks.

2. DESIGN OVERVIEW

Current distributed framework design is inappropriate in accommodating component applications with numerous components that use many computing resources. We implemented a CCA-compliant distributed component framework called CCALoop that prototypes our design for increased framework scalability and fault tolerance. CCALoop scales by dividing framework data storage and lookup responsibilities among its nodes. It is designed to provide fault-tolerance and uninterrupted services on limited framework failure. CCALoop also provides the ability to connect multiple GUIs in order for users to monitor an application from multiple points. While providing these capabilities CCALoop does not add overwhelming overhead or cost to the user and satisfies framework queries with low latency. In this section we will examine the parts that form the structure of this framework. We begin by looking more closely at the tasks and roles of a CCA-compliant component framework.

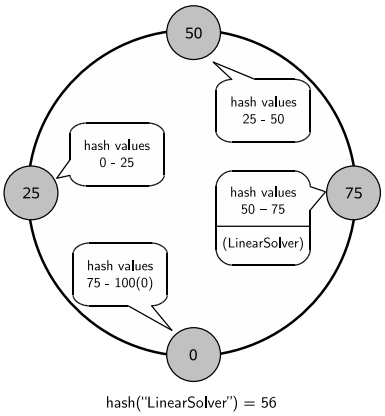

Figure 1: The data responsibilities of the CCALoop framework with four nodes.

The main purpose of a component framework is to manage and disseminate data. Some frameworks are more involved but in this work we focus on the ones in the style of CORBA [6] that do not interfere with the execution of every component. This kind of a component framework performs several important tasks in the staging of an application, but gets out of the way of the actual execution. Executing components may access the framework to obtain data or to manage other components if they choose to, but it is not usually necessary. CCA-compliant frameworks also follow this paradigm as it means low overhead and better performance.

CCA-compliant frameworks store two types of data: static and dynamic. The majority of the data is dynamic, which means that it changes as the application changes. The relatively small amount of static data describes the available components in the system. In a distributed setting, static data consists of the available components on each distributed framework node. The dynamic data ranges from information on instantiated components and ports to results and error messages. A significant amount of dynamic data is usually displayed to the user via a GUI. In our design, we distribute management of framework data without relocating components or forcing the user to instantiate components on a specific resource. The user is allowed to make his or her own decisions regarding application resource usage.

One of the principal design goals of CCALoop is to balance the load of managing component data and answering queries to all participating frameworks. This is done by using a DHT mechanism where each node in the system is assigned a unique identifier in a particular identifier space. This identifier is chosen to ensure an even distribution of the framework identifiers across the identifier space. We provide an operation that hashes each component type to a number in the identifier space. All metadata for a given component is stored at the framework node whose identifier is the successor of the component hash as shown in Figure 1. Given a random hash function the component data is distributed evenly across the framework nodes. The lookup mechanism is similar to the storage one: to get information about a component, we compute its hash and query the succeeding framework node.

CCALoop's framework nodes are organized in a ring structure in topological order by their identifier numbers. Each framework node has a pointer to its successor and predecessor, allowing the ring to span the identifier space regardless of how the system may change or how many nodes exists in a given time. CCALoop also facilitates a straightforward way of recovering from node failure, by naturally involving the successor of the failed node to become new successor to the queried identifier.

3. REFERENCES

[1] 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*, 14(5):323–345, 2002.

[2] 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 the 8th IEEE International Symposium on High Performance Distributed Computation (HPDC)*, August 1999.

[3] S. Gribble, E. Brewer, J. Hellerstein, and D. Culler. Scalable, distributed data structures for Internet service construction. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, 2000.

[4] S. Krishnan and D. Gannon. XCAT3: A framework for CCA components as OGSA services. In *Proceedings of The 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, April 2004.

[5] L. C. McInnes, B. A. Allan, R. Armstrong, S. J. Benson, D. E. Bernholdt, T. L. Dahlgren, L. F. Diachin, M. Krishnan, J. A. Kohl, J. W. Larson, S. Lefantzi, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, and S. Zhou. Parallel PDE-based simulations using the Common Component Architecture. In A. M. Bruaset and A. Tveito, editors, *Numerical Solution of PDEs on Parallel Computers*, volume 51 of *Lecture Notes in Computational Science and Engineering (LNCSE)*, pages 327–384. Springer-Verlag, 2006.

[6] OMG. *The Common Object Request Broker: Architecture and Specification. Revision 2.0*, June 1995.

[7] K. Zhang, K. Damevski, V. Venkatachalapathy, and S. Parker. SCIRun2: A CCA framework for high performance computing. In *Proceedings of The 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, April 2004.