DOGMA: An Open Source Tool for Utilization of Idle Cycles on Lab Computers

Nathan H. Ekstrom, Joseph J. Ekstrom
Brigham Young University

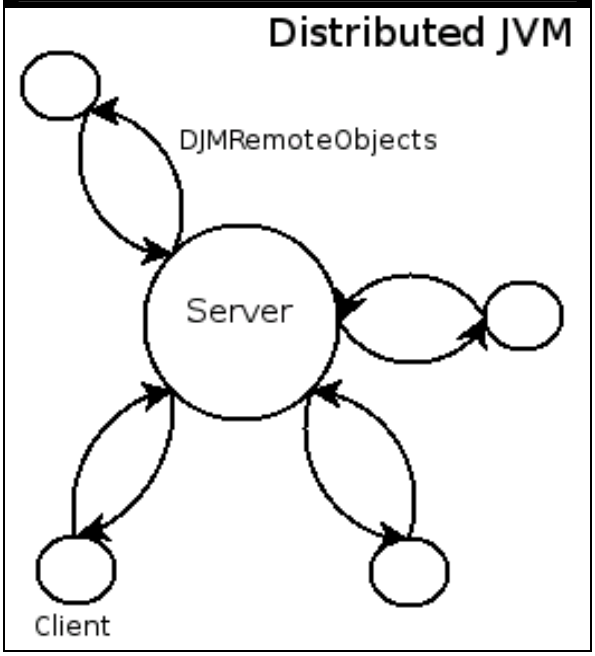
Abstract
Organizations often have many computers that are unused for much of the day. The desire to utilize these idle machines has spawned systems that attempt to harness the unused computer cycles for useful work. These include SETI, Globus, Condor, DOGMA, and recently SLURM. In the late 1990's the Distributed Object Group Management Architecture (DOGMA) project was begun in the Network Computing Lab in the Computer Science department at Brigham Young University. DOGMA is a Java based system that allocates Java programs (jobs) to unused workstations. Although DOGMA currently has over 700 desktop workstations available for use overnight, there were several issues which impeded wide acceptance. These included robustness of the implementation, maintainability, and management issues. Many of these issues have been overcome in the most recent implementation.

This paper will discuss DOGMA including its basic design and the current status of the project. We will also discuss alternatives for its future evolution. It is interesting to observe that many of the unresolved issues are of little interest as Computer Science problems but may be of great interest to Information Technology researchers.

Introduction

Organizations often have many computers that are unused for much of the day. The desire to utilize these idle machines has spawned systems that attempt to harness the unused computer cycles for useful work. These include SETI@home [1], Globus[2], Condor[3], DOGMA[4], and recently SLURM[5]. Each of these systems has unique characteristics and has evolved independently. At BYU DOGMA is being used to facilitate phylogentic research and 3D graphics rendering. These compute intensive applications are being executed at no cost through the use of idle cycles. This enables even undergraduates to have access to extensive computing resources.

Figure 1



The goal of this paper is to trace the evolution of DOGMA rather than to analyze the relationships between these systems.

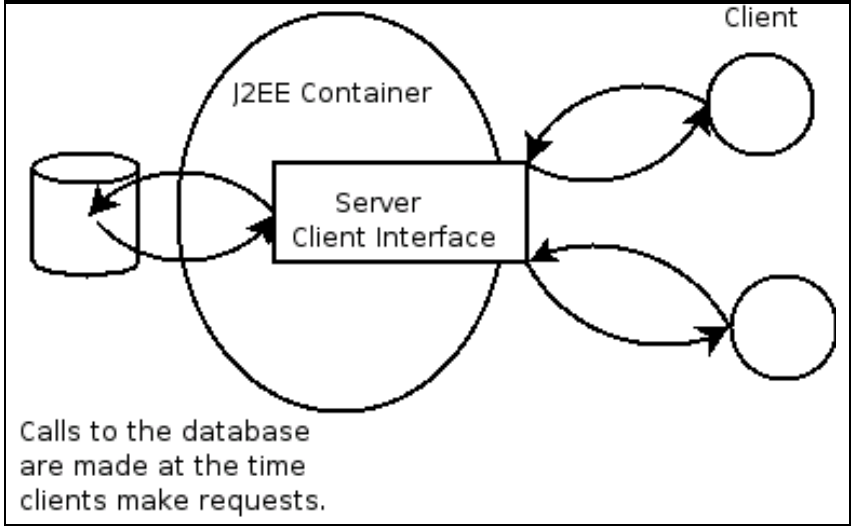
In the late 1990's the Distributed Object Group Management Architecture (DOGMA) project was begun in the Network Computing Lab in the Computer Science department at Brigham Young University. DOGMA is a Java based system that allocates Java programs (jobs) to unused workstations. Although DOGMA currently has over 700 desktop workstations available for use overnight, there were several issues which impeded wide acceptance. These included robustness of the implementation, maintainability, and management issues. Many of these issues have been overcome in the most recent implementation.

The Distributed Object Group Management Architecture (DOGMA) has gone through several implementation cycles. The original concept was as a toolkit for distributed computing across the Internet using Java. This toolkit would allow the creation of a "Distributed Java Machine" (DJM) as illustrated in figure 1. Nodes that wished to participate would launch a node manager which in turn would interact with a central server. A program run on this system would then make calls to functions using DJMRemoteObjects; these are special RMI Server objects which instead of having the code run locally send the function request to the central server which then sends the request to an open node. The node processes the request and returns the results, just as a normal function would. A test application was made for this system that did large matrix multiplication. The system worked and plans were made for future research.[4]

The second implementation of DOGMA was an attempt to separate the scheduler from the distributed Java machine. This would allow the running of multiple programs on the system at once, something the original version did not support. Instead of working with the old code base however, and separating the pieces, the graduate student doing the work decided to do a complete rewrite. His project never reached a usable state.

The third version of DOGMA, called DOGMA-NG or DOGMA Next Generation, was an attempt to create a usable resource scheduler that could interact with idle compute resources as well as super computer schedulers. It was written using J2EE technology and ran on the JBoss 3.0.8[6] open source J2EE

Figure 2



container. Figure 2 illustrates this architecture. The deployed Enterprise Java Beans[7] (EJBs) acted as a scheduler. This version never supported enough clients to be widely used. With a small number of machines the system worked fine; however when it was deployed with around one hundred clients the system's ability to respond to requests quickly deteriorated causing the clients to time out. Eventually the server would crash.

The current version of DOGMA, termed DOGMA3, is a total rewrite of the DOGMA-NG system using newer tool technology. Like DOGMA-NG it harnesses Java Enterprise technology. However, the development effort was significantly reduced by using more current tools. These include Middlegen[8] and XDoclet[9] which allow for rapid creation of Enterprise Java Beans. XDoclet allows one to create one java file for each entity bean you wish to create. The developer specifies what other files should be generated as well as what functions should be in them through the use of special JavaDoc tags. An Ant[10] build task is then used to generate the other files. Middlegen is a tool that will connect to a database and then generate java files which already have the XDoclet tags embedded in them. By using both of these technologies half of the code was automatically generated instead of having to be written by hand.

DOGMA3 has been delivered. It is currently being used to harness the idle cycles of machines across the Brigham Young University campus. It has handled over 800 clients at one time and has exhibited none of the older version's stability problems.

There are several areas of DOGMA3 that could be improved, including, client side error reporting, precise and specific statistics gathering of the client and server, scalability beyond 800 nodes, and network usage. Graduate students are currently working two of these improvements while the others, being of little interest to researchers in the Network Computing Lab, are being ignored at least for the present.

DOGMA3 Design

The DOGMA3 system is a scheduler for distributed programs. Its primary function is to allow for programs to be launched on a client machines. Normally, files necessary for a given application to run are first copied to the client node. A command line is provided for the client to launch the program. After the launched program exits its return status is reported to the server and the node requests another program for execution.

DOGMA3 uses a client server architecture in which all communication is initiated by the client. One of the difficulties associated with a system of this type is how to propagate upgrades and changes to the clients in an efficient manner. DOGMA3 uses a two piece client to accomplish this. The first piece is actually a very simple bootstrap client whose job is to launch the real client which actually communicates with the server and runs the desired programs. The bootstrapper consists of a small Java Jar file and a Java class repository. The Jar file is launched and uses the class repository, located on a server available through HTTP, to download all of the files needed by the second part of the client. Once all of the files are downloaded the main client is launched and the node reports in and asks for work. The client is always using current code because the class

repository on the server can be easily updated and it always downloads the newest files for the main client.

The main client, referred to hereafter as the client, consists of a main loop and launched processes. The main loop consists of some simple logic which sends a heartbeat to the main server. Included in this heartbeat is a list of what tasks, if any, the client is currently running. In response the clients is told whether it should keep working on its current tasks or if it should stop those and start others.

Associated with the client are several helper classes which provide communication to and from the main server as well as download and upload capabilities. These classes are abstracted and it is possible for any class which implements the necessary interfaces to be used instead.

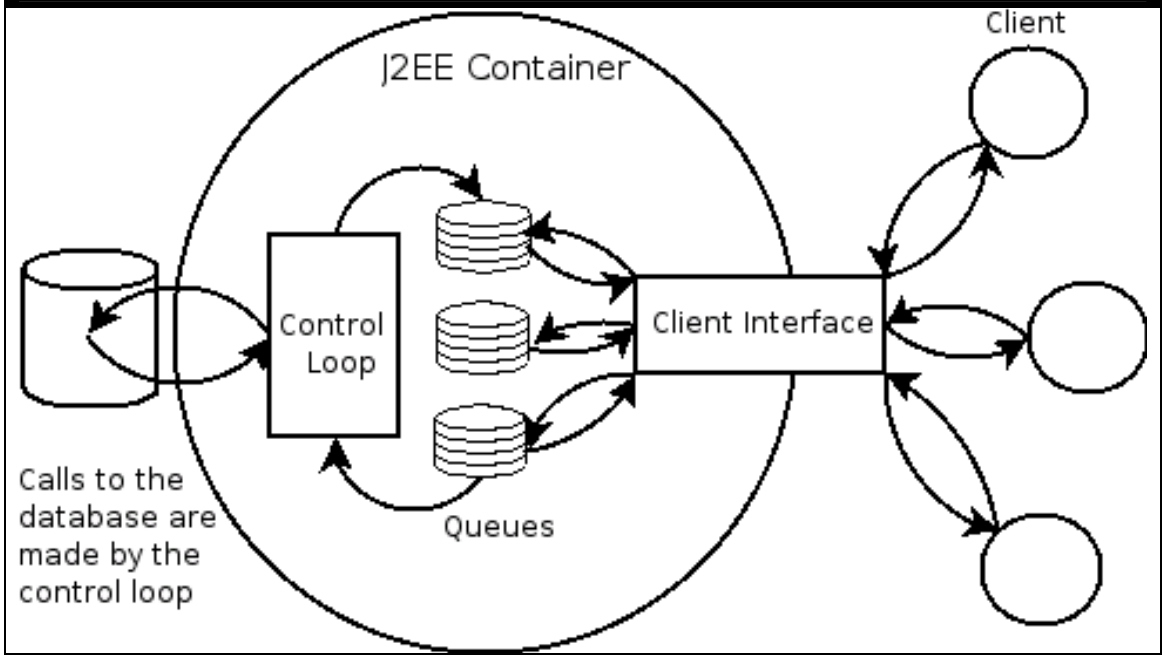
The DOGMA client also uses classes run in separate threads to launch programs designated by the server. These classes implement an interface and can thus be easily replaced by other classes which implement the same interface. Currently there are two versions of this class. One version is used to run command line jobs which behave the same as if you were sitting at the console and typing the command. The second version is used to run Java class files from inside the same Java Virtual Machine as the client.

For the Java class file to be run it must implement a specific interface. Through this interface information is given to the class regarding its current task, as well as access to communication classes which provide SOAP [11] communications, HTTP upload and download capabilities, and task error reporting. Because this class is run inside the client's class loader it also has access to all of the client's classes and libraries as well as those downloaded and specified by the job description.

The DOGMA3 server runs in the JBoss 3.2.x [6] Enterprise Java Bean Container. At the heart of the server are several control threads and a client communication interface. As described in figure 3, the clients communicate with the server using SOAP objects. The JBoss system uses Apache Axis[12] to expose desired functions as SOAP RPC.

One of the major drawbacks of DOGMA-NG was its inability to scale. This was partially due to the fact that every time a client requested work several database calls were made to decide what work was available and what to give the client. To remove this bottleneck DOGMA3 makes use of caching and in-memory queues. Every few seconds control threads run which: commit information received from clients to the database, look at current client information to decide which clients are still active, and finally update and fill scheduling queues which are used to assign work to clients when they check in. By making use of these structures and reducing the number accesses to the database, DOGMA3 is currently able to handle around 800 nodes checking in every two to three minutes.

Figure 3

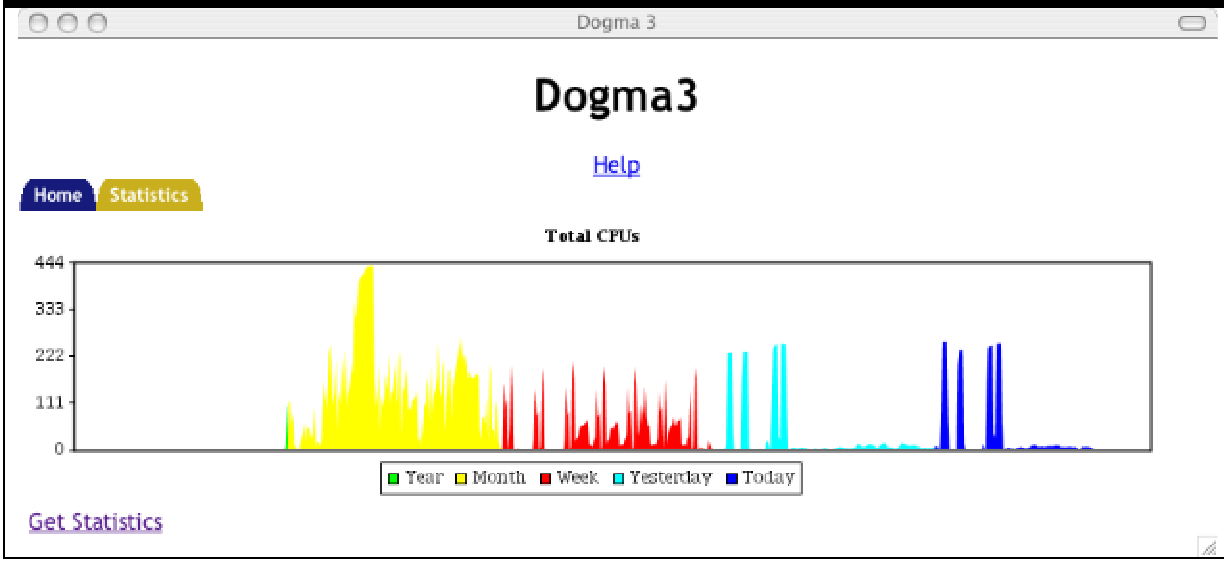


Another problem of the DOGMA-NG server was the interdependence of the scheduler and the user interface. This architecture was difficult to understand and maintain. With DOGMA3 the scheduler and the user interface have been separated and can each run independent of the other. This architecture provides for independent evolution of the components.

To allow for easier interaction of other systems with DOGMA, DOGMA3 exposes web services interface. These web services have greatly simplify the task of creating programs that run over DOGMA and are meant to be the main interaction point with the DOGMA system.

A new web-based interface was created to allow for easy interaction with DOGMA. In addition a change was made from a simple two level security system to a privilege based architecture that supports additional use models. Graphs, such as figure 4 (which is a time compressed graph of the total CPUs available to the system), and better job monitoring were also added to enable management of monitoring of the system.

Figure 4



Problems and Future Research

The outstanding issues with DOGMA3 are less interesting to NCL researchers than the original projects. Error reporting is a critical part of any robust system. Currently DOGMA3 has only basic error reporting done by the clients involving programs being run and not the client itself.

Statistics are also important when using a system of this type and though some statistics, such as number of CPUs available for use and how many CPUs a job is currently using, are kept, further statistics involving individual clients and perhaps more specific statistics involving jobs would be useful. These statistics could be used to make better scheduling decisions for jobs as well as allow for requirements to be imposed on clients for certain jobs, such as a minimum average time the client is available.

Another problem is scalability. DOGMA3 currently makes use of queues and caching to help increase the number clients a single server can support. A dual 2.4GHZ Xeon with 2 GB of ram running the DOGMA3 server can support around 800 clients. This is a fairly powerful machine, and while an even more powerful machine should be able to handle more clients we are interested in looking at ways to reduce the per-CPU overhead for the server. One possibility is the dynamic creation of peer cluster groups.

A peer cluster group consists of clients within the same broadcast domain. When clients first start up they attempt to locate a peer cluster to join; if they cannot find one they start a new one. Each peer cluster elects a leader, usually the client who started the cluster; all communication with the server is then done by the leader. The cluster of machines then acts and reports to the main server as a single machine with many CPUs; this *virtual machine* is as powerful as all of the machines in the cluster but has a much lower central server overhead than communication with each of the individual machines.

Another problem DOGMA faces is network bandwidth constraints. Every client downloads every file for every job it runs from a central location set during job creation. If a job has many parts and many files, significant network load is created and a bottleneck occurs if the files are accessed over a slow connection. Consider a job that requires a one megabyte download before it can be run: if the job is allocated to 500 clients 500 megabytes will be requested as the job starts each of its parts. A bottleneck now exists if the server providing the files is on a slow connection.

To help alleviate this problem functionality is being added to the clients which will allow them to query clients in the same broadcast domain for a specific file. This will allow clients to acquire files from local sources to which they have a much faster connection.

Current Open Source Version

DOGMA3 has been made an open source project under the GNU Public License version 2. This allows for interested researchers and individuals to contribute to the project. Hopefully this will provide for future development and support by other groups besides the Networked Computing Lab, whose research interests are centered on specific parallel algorithms rather than development of distributed job management infrastructure.

Maintenance is always an issue with any project. Historically, researchers loose interest after a project has reached a usable state, because the problems that interested them have been shown to have solutions by the existing system. Maintenance and system evolution is often not an interesting research problem to the original team. It is hoped that giving DOGMA to the Open Source community will allow its use by a broader community and provide for its evolution by interested parties.

Lessons Learned and Outstanding Issues

As technology and standards advance, as well as available tools, it is often beneficial to evolve projects with the tools. At times this may require that a project be rewritten from scratch, as was the case with DOGMA. Though the principal researchers were at first reluctant to do another rewrite, it has proven beneficial for DOGMA's maintainability, robustness, and performance.

Though DOGMA has evolved to overcome many of the problems which have impeded its adoption, there are still issues that still need to be addressed. These include specific statistics gathering for the client and server, better client error reporting, network usage monitoring, and scaling for number of clients and bandwidth required to service large jobs from a central service point.

Conclusion

The DOGMA3 system provides a useful platform upon which to implement certain classes of distributed algorithms. It provides an infrastructure that can allow researchers to use idle workstations for computations that formerly required a supercomputer. The current version has been released under the GPL2 Open Source license. This should allow a much wider audience to benefit from this infrastructure. It is especially

appropriate for academic institutions that have large numbers of open-access lab workstations.

As the authors have discussed the evolving status of DOGMA over several years, it has become clear that many of the emerging issues are of little interest to the researchers who originated the project. However, the issues of evolving systems over time, incremental scaling, and component monitoring and management are very interesting to many Information Technology professionals and managers. As the academic discipline of Information Technology has emerged over the last few years[13], many have asked how the research agenda of IT relates to that of Computer Science, Computer Engineering, and Information Systems. Recent drafts from The Joint IEEE Computer Society/ACM Task Force on the "Model Curricula for Computing" for CC2004 include an overview document [14] that describes the relationship between these sister computing disciplines. The center of Computer Science as a discipline is the study of algorithms, programming, and languages and systems that allow one to program and evaluate implementations of algorithms. The center of Information Technology is system integration and deployment of systems into a user community.

Using the evolution of DOGMA as an example, one can hypothesize that there is a natural evolution of projects from problems of interest to Computer Science researchers to problems of more interest to Information Technology researchers. It is also possible that the Open Source model provides a natural transition mechanism that can be of benefit to the entire academic community as interesting projects are adopted and deployed into larger constituencies.

REFERENCES

[1] Anderson, David P., Cobb, Jeff, Korpela, Eric, Lebofsky, Matt, Werthimer, Dan, *SETI@home: an experiment in public-resource computing*, Communications of the ACM Volume 45, Issue 11, Pages 56-61, 2002, ISSN:0001-0782

[2] Foster, Ian, Kesselman, Carl, *Globus: A metacomputing infrastructure toolkit*, International Journal of Supercomputer Applications, 11(2):115-128, 1997

[3] Thain, Douglas, Tannenbaum, Todd, Livny, Miron, *Distributed Computing in Practice: The Condor Experience*, Concurrency and Computation: Practice and Experience, 2004

[4] Judd, G., Clement, Mark, Snell, Quinn, *DOGMA: Distributed Object Group Management Architecture*, http://ncl.cs.byu.edu/publications/ps/dogma.ps

[5] Jette, Morris, Grondona, Mark, *SLURM: Simple Linux Utility for Resource Management*, submitted to ClusterWorld Conference and Expo, June 23, 2003

[6] http://www.jboss.org, January 2, 2005

[7] http://java.sun.com/products/ejb/index.jsp, January 3, 2005

[8] http://boss.bekk.no/boss/middlegen, January 2, 2005

[9] Walls, Craig, Richards, Norman, *XDoclet in Action*, Manning Publishing Co., Greenwich, CT, USA, ISBN 1932394052

[10] http://ant.apache.org, January 3, 2005

[11] Clements, Tom, *Overview of SOAP*, http://java.sun.com/developer/technicalArticles/xml/webservices/ January 2002

[12] http://ws.apache.org/axis, January 3, 2005

[13] Lunt, Barry, et al., *Defining the IT Curriculum: The Results of the Past 2½ Years*, ASEE 2004, Salt Lake City, Utah, June 2004.

[14] CC2004 Overview Report, http://www.acm.org/education/Overview_Draft_11-22-04.pdf , retrieved January 3, 2005

JOSEPH J. EKSTROM

Joseph J. Ekstrom (Ph. D. Computer Science, BYU 1992) has been Associate Professor of Information Technology at BYU since 2001. During 30 years of industrial experience he held positions from developer through senior management. His research interests include network and systems management, distributed computing, system modeling and architecture, system development, and IT curriculum and instruction.

NATHAN H. EKSTROM

Nathan H. Ekstrom (BS Computer Science, BYU 2004) is a graduate student working in the Networked Computing Lab of the Computer Science Department at BYU and is the principal author of DOGMA3. His research interests include distributed systems, enterprise systems, and enterprise computing management.