Agent Space Architecture for Search Engines

Ben Choi & Rohit Dhawan

Computer Science, College of Science and Engineering
Louisiana Tech University, LA 71272, USA
pro@BenChoi.org

Abstract

The future of computing is moving from individual processing units to communities of self organizing agents. In this paper we propose a new agent and network based architecture for parallel and distributed computing called Agent Space Architecture. Our architecture builds upon the notions of agent and Object Space and utilizes multicast networks. The building blocks for our proposed architecture consist of an active processing unit called agent, a shared place for communication called space, and a communication medium called multicast network. One unique feature of our architecture is that we extend the concept of Object Space to become an Active Space. Our Active Space functions as a rendezvous, a repository, a cache, a responder, a notifier, and a manager of its own resources. The organization of our architecture is as general as network topology. Any number of agents, spaces, or networks can be added to achieve high performance. It is as scalable as Ethernet and adding agents or spaces is as easy as plug and play. High availability and fault tolerance is achieved through multiple agents, spaces, and networks. All these features are particularly beneficial for challenging applications such as search engine, which is used as a test case to implement and to test our proposed architecture.

1 Introduction

Parallel and distributed computing has great potential for exploiting the vast computational power of millions of personal computers all over the world. Although there are successful cases of using large number of PC's, the current difficulty for scalability is due largely to the highly coupling on the underlying management software and parallel programming interfaces such as Message Passing Interface (MPI). For instance, Google architecture utilizes over 15000 PC's and continues to adding more for keeping up with the explosive growth of the number of Web pages [2-4]. It also utilizes separated fault tolerance software [4] and MPI, which make management, administration, and configuration of such a large server farm become a major issue. The search engine architecture of Inktomi Corporation [1,7] serves portals such as Yahoo, HotBot, Microsoft MSN, Geocities, NTT "goo" Tokyo. It is a cluster based architecture utilizing RAID arrays [6] and Myrinet [14]. AltaVista, Lycos, and Excite make use of large SMP supercomputers [1] and as such fault tolerance is done through multiple replicated SMP, which results in limited scalability but costly replications. Search engine represents a challenging application for parallel and distributed processing, which is used as a test case to implement and to test our proposed architecture.

In this paper we propose a parallel and distributed computing architecture that is highly modular and requires least human intervention. Our architecture builds upon the notions of agent and Object Space and utilizes multicast networks. The building blocks for our proposed architecture consist of an active processing unit called agent, a shared place for communication called space, and a communication medium called multicast network. Multiple building blocks are organized to form a parallel and distributed computing architecture.

2 Related Research

Our proposed architecture is built upon an extended notion of Object Space [5]. An Object Space is a shared medium that simply acts as a rendezvous for agents to meet there either to serve or be served without the knowledge of each others identity, location, or specialization. Other variations of Object Space are JavaSpace [5], IBM's TSpaces [10], TONIC [11], JINI [15], and TupleSpace [12, 16].

Several architectures based on the notion of Object Space have been proposed. One of the proposed architecture [8] utilizes an Object Space as a repository of various roles where agents adapt to changing demands placed on the system by dynamically requesting their behavior from the space. A framework for cluster computing using JavaSpace [5], Object Space for Java, has been described in [9].

This research was supported in part by a grant from the Center for Entrepreneurship and Information Technology (CEnIT), Louisiana Tech University.

It uses a network management module for monitoring the state of the agents and uses the state information to schedule tasks to the agents. JavaSpaces has also been used for scientific computation [13]. It supports the perspective that JavaSpace solution has merits when used in high performance computing.

3 Our Proposed Agent Space Architecture

The building blocks for our proposed Agent Space Architecture consists of an active processing unit called agent, a shared place for communication called space, and a communication medium called multicast network. Multiple building blocks can easily be organized to form a parallel and distributed computing architecture.

3.1 Agent

Our agent plays many active roles in parallel and distributed computing. In the concept of Object Space, our agent can function as a master or a worker. In general, our agent is an active processing unit. Once started it actively seeks for tasks to be done and actively monitors events to be handled.

Multiple agents work in groups. Agents can join or leave a workgroup. Agents within a workgroup communicate with each other through shared spaces. If an agent needs a certain task to be done by other agents, it can simply deposit the task into a shared space. Another agent monitoring the space can pick up the task, complete the task, and deposit the results back into the space. Then, the original agent can simply pick up the results from the space.

An agent who needed a task to be completed only cares the task eventually got done. It does not care who completed the task. In our Agent Space Architecture, an agent does not even know the identity of other agents. This creates a working environment in which agents can join or leave with ease. However, this also creates a working environment in which an agent does not have the assurance that its task will eventually get done. One way to have such an assurance is to insure at least one agent that can complete the task remains in the workgroup. The abilities of a workgroup are in part the sum of the abilities of its agent workers.

3.2 Space

Our space extends the concept of Object Space to become an Active Space. Our Active Space functions as a rendezvous, a repository, a cache, a responder, a notifier, and a manager of its own resources. It acts as a rendezvous for Agents to communicate with each others. It acts as a repository and a share memory for agents to deposit requests and temporally save results. It acts as a cache in that results are stored there and when an agent needs the results it simply reads the results. This reduces repeated computations when several agents need the same results.

Our Active Space also acts as a responder to let agents know that it is there, when an agent is trying to discover a space to join. In addition, it acts as a notifier to broadcast its state changes for events such as, new requests are deposited or new results are returned.

The Active Space also acts as a manager of its own resources. It is a manager for its rendezvous, repository, cache, responder, and notifier. For instance, it manages its memory as cache manager that takes care of cache replacements.

3.3 Multicast Network

We select multicast network as a medium for our agents to communicate with our Active Spaces and for the Active Spaces to broadcast events to agents. Multicast network [17, 18] provides the facilities for agents and spaces to function in a group. An agent joins the group by simply connecting to the network and announces its present. Active Spaces in the network will respond to the newcomer. This establishes direct communication between the agent and the spaces.

3.4 Organization for Parallel and Distributed Computing

The organization of our parallel and distributed computing architecture is as general as network topology. The minimum functional configuration requires one agent, one space, and one multicast network. This configuration can easily be expanded by simply connecting more agents or more spaces into the network. It is as easy as plug and play. Once connected, an agent will automatically discover spaces to join, and a space will automatically announce its present and invite agent to join the new space. The organization can further be expanded by adding more networks. An agent or a space can connect to multiple networks (or network segments) by using multiple ports.

4 Design for Search Engine Application

Although our Agent Space Architecture is a general framework for parallel and distributed computing, we choose search engine application as a test case for implementing and testing our architecture. We have implemented and tested our Search Engine Architecture (Figure 1) [21]. To reduce development time we implemented our architecture by utilizing

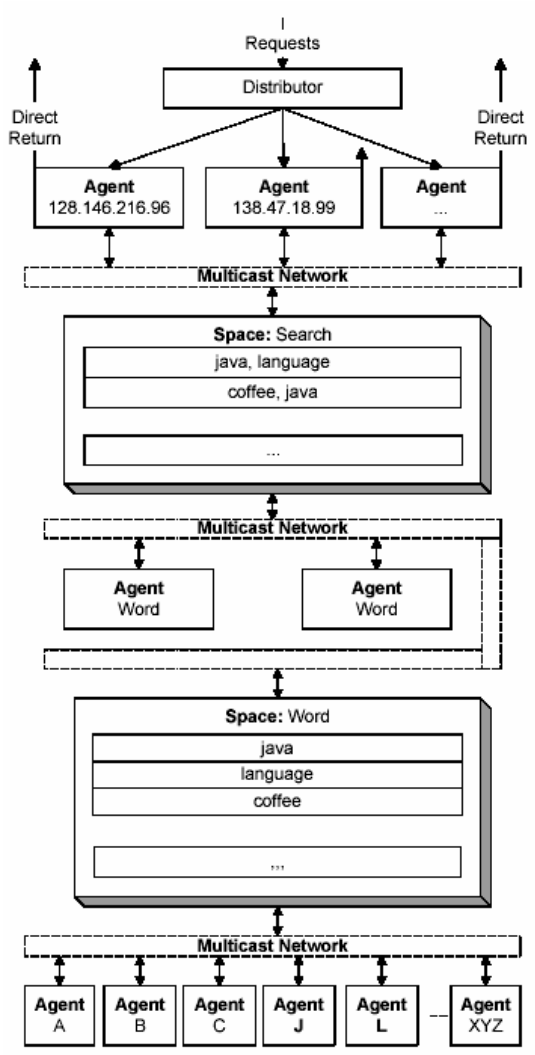

Figure 1. Agent Space Architecture for Search Engines

API's of JINI platform provided by Sun Microsystems [15] and by using Apache and Tomcat as our web and application servers.

Implementing Agents

All our agents automatically discover a space to join as soon as they are started. After joining a space, an agent becomes a member of the workgroup specified by the name of the space. For instance, as shown in Figure 1, the agents in the bottom row, Agent A, Agent B, and so on, are members of workgroup Space Word. An agent can be a member of multiple workgroups. For instance, each of the Agent Word in Figure 1 is a member of workgroup Space Word and a member of workgroup Space Search. We implemented this automatically discovery process for our agent by utilizing the Discovery and Lookup

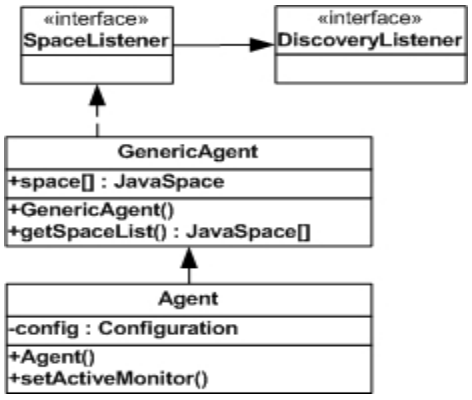

Figure 2. An Agent

protocols provided by JINI [15]. In particular, as shown in Figure 2, we use a SpaceListener interface that extends the DiscoverListener interface defined in JINI. During the discovery process, an agent broadcasts a request through a multicast network. Active Spaces, such as Space Search or Space Word in Figure 1, actively monitoring the multicast network will respond to the request. As soon as the agent receives a response from a space, it becomes part of the workgroup.

All our agents also automatically monitor events to be handled within their workgroups. The events include new tasks need to be completed or new results returned. An event is implemented as a type of *net.jini.core.event.RemoteEvent* provided in JINI API. To reduce communication overhead, we further divide events into to types: group events and individual events. Group events, such as new tasks needed to be handled, are broadcast to all members of the workgroup. Individual events, such as new results that an agent is waiting for have returned, are sent to the particular agent waiting for the events.

Different types of agents are able to handle different types of tasks. For example, as shown in Figure 1, those agents directly below the distributor are able to handle search requests and to send the search results directly to the clients, while those agents in the bottom row of Figure 1 are able to search database for web pages matching the requested keywords. To capture the common characteristics of agents and the individual abilities of different types of agents, we implemented each of our agents as defined in Figure 2. The individual abilities of different types of agents are specified by using the attribute called config that has the type Configuration defined in net.jini.config.Configuration of the JINI API.

Implementing Spaces

To implement and test our Active Spaces, the Space Search and the Space Word for our Search Engine Architecture (Figure 1), we utilized the Lookup service and the JavaSpace service provided by Sun Microsystems [15]. Our Active Space automatically responds to agents during the process for the agents to discover spaces to join. This is made possible through the Lookup service and the multicast networks (to be described). The Lookup service monitors multicast requests and helps establish direct communication between an agent and a space.

Our Active Space functions like a repository for agents to write, read, or take (that is read and remove) tasks, for which we simply used the corresponding functions provided by JavaSpace service. We also make our Active Space functions like a cache, in which the results of a task are stored for a specific duration. When an agent needs to complete certain task and finds that the results of the task are already available, it will simply retrieve the results. We simulated the first-in-first-out cache replacement policy by using the Lease function provided in JINI API. After a lease has expired, the results of a task will be removed from the space.

When a new task is written into our Active Space, the space will broadcast this event to all worker agents who are listening to the space. This function is implemented by using the notify method available in JavaSpace service. In addition, we also use the notify method for a space to notify a particle agent when the results that the agent is waiting have been written by other agent.

Implementing Multicast Networks

We implemented our multicast networks by using three protocols, multicast request protocol, multicast announcement protocol, and unicast protocol, which are provided in JINI API. Multicast request protocol is used by agents to discover Active Spaces via Lookup service. Multicast announcement protocol is used by Active Spaces to announce, via Lookup Service, their presence in the network. And, unicast protocol is used to establish direct communication between agents and spaces after the discovery process has been completed.

In our implementation, we simulated the three different multicast networks as depicted in Figure 1 by using one ordinary Ethernet. This simulation does not allow partitioning of network traffics, but is sufficient for our testing purpose. An agent is preset to discovery certain spaces. For example, the agents in the bottom row of Figure 1 are preset to discover Space Word, while an Agent Word is present to discover both Space Word and Space Search.

Implementing Distributor

The distributor receives requests from clients and distributes the requests to be processed by agents as depicted in Figure 1. For our testing purpose, we used Apache web server to receive requests from clients and to pass the requests to Tomcat application server. We implemented the distributor by using a Java Server Page running within Tomcat and redirecting the requests to the underlying agents. For load balance, the distributor redirects a request to an agent that is least recently used.

5 Benefits of Our Proposed Agent Space Architecture

Our Agent Space Architecture provides a general framework for parallel and distributed computing. Supported by our test results, the proposed architecture addresses major issues including high performance, high scalability, ease of management, high availability, and fault tolerance.

High Performance

High performance of our proposed architecture is achieved by simply adding more agents, spaces, or networks. Another feature of our architecture for high performance is the result of using space as cache. When an agent needs to perform certain task and finds that the result of the task is already stored in the space, there is not need to repeat the computations. The agent simply reads the results from the cache. This not only reduces repeated computations when several agents need the same results but also reduces the response time, which is practically beneficial for search engine applications.

High Scalability and Ease of Management

Our architecture is as scalable as Ethernet. Any number of agents, spaces, or networks can be added. Adding an Agent is as simple as connecting the agent to a network. The agent will then discover a space in the network and become part of the workgroup. It is a plug and play process. No manual configuration is needed. Similarly, adding a space is simply connecting the space to the network and the space will broadcast its present through the multicast network. Adding a network is as easy by connecting agents and spaces into the network. This is made possible by the fact that agents and spaces can be connected to multiple networks through multiple ports.

High Availability and Fault Tolerance

High availability and fault tolerance is achieved through multiple agents, spaces, and networks. For instance, having multiple agents performing the same role, the failure of an agent only downgrade the performance and will not affect the overall functionality of the system. Replacing an agent can be as simple as disconnecting the agent from the network and connecting another one. Having multiple spaces, the failure of one space again will only downgrade the performance. An agent pending for a request to be completed will discover that the space is not available and will then send the request to another space. Similarly, having multiple networks, the failure of one network will only downgrade the performance in a larger extent. Agents and spaces can continue to communicate through their ports that are connected to a live network.

6 Conclusion and Future Research

Our highly distributed, hierarchical, modular architecture promises high performance, scalability, and availability, requires less human intervention, and provides natural fault tolerance. Our experiences with the architecture indicate that such a system is easily configurable, extensible and hence mitigates the management issues confronted by existing search engine architectures. Although we implemented and tested our proposed Agent Space Architecture for search engine application and in particular used to run our Information Classification and Search Engine [19-22], the proposed architecture is general enough for other high demand applications that require parallel and distributed computing.

The framework outlined in this paper by using agents and by using shared space for communication among the agents can be applied to form computing communities where agents come and leave at will. The power of a computing community may be more the sum of its parts. The future of computing is moving from individual processing units to communities of self organizing agents.

References

[1] Eric A. Brewer, "Inktomi Architecture, UC Berkley," http: //www.acm.org/ sigs/ sigmod/ disc/ disc99/ disc/nsf_acad_ind/ brewer/ index.htm.

[2] Sergey Brin and Lawrence Page, "The anatomy of a large-scale web search engine", Proceedings of the 7th Intl. WWW Conf., 107-117, 1998.

[3] Intel Corporation, Google, "http: //www. intel.com/ eBusiness/casestudies/snapshots/ google.htm"

[4] Mitch Wagne, "Google defies Dotcom DownTurn," http:// www. Internetwk.com/ story/ INW20010427S0010

[5] Eric Freeman, Susanne Hupfer, and Ken Arnold, JavaSpaces: Principles, Patterns, and Practice, Addison-Wesley, Reading, Massachusetts, 1999.

[6] Vijay Karamcheti, "Scalable Clusters: Architecture and Software" http://www.Cs.nyu.edu/ courses/ spring98/ G22.3033.10/ lectures/lect0414.pdf, 1998.

[7] Inktomi, http://inktomi.com

[8] Engelhardtsen and Gagnes, "Using JavaSpaces to create adaptive distributed systems", http://www.nik.no/ 2002/ Engelhardtsen.pdf , 2002.

[9] Batheja and Parashar, "A Framework for Opportunistic Cluster Computing using JavaSpaces", http: //www. caip. rutgers. edu/ TASSL/ Papers/ jinihpc-hpcn01.pdf , 2001

[10] T.Lehman et al, IBM Almaden research Center, http://www.almaden.ibm.com/cs/TSpaces/

[11] TONIC, "Scientific Computing with JAVA TupleSpaces", http://hea-www.harvard.edu/ ~mnoble/tonic/doc/

[12] OpenWings, "Service Oriented Architecture", http://www.openwings.org.

[13] Michael S. Noble and Stoyanka Zlateva, "Scientific computation with javaspaces," in Proceedings of the 9th International Conference on High Performance Computing and Networking, June 2001.

[14] Myrinet, http://www.myrinet.com

[15] JINI, "Jini Specifications and API Archive", http://java.sun.com/products/jini/

[16] Nicholas Carriero, David Gelernter: A Computational Model of Everything. CACM 44(11): 77-81, 2001.

[17] Su Wen, James Griffioen, and Kenneth Calvert. Building multicast services from unicast forwarding and ephemeral state. In OPENARCH 01, March 2001.

[18] Beau Williamson, Developing IP Multicast Networks, Vol. 1, Cisco Press, 1999.

[19] Ben Choi, "Making Sense of Search Results by Automatic Web-page Classifications," Proc. of WebNet 2001 -- World Conference on the WWW and Internet, pp.184-186, 2001.

[20] Ben Choi and Xiaogang Peng "Dynamic and Hierarchical Classification of Web Pages," Online Information Review, Vol. 28, No. 2, pp. 139-147, 2004

[21] Ben Choi and Rohit Dhawan, "Distributed Object Space Cluster Architecture for Search Engines," High Availability and Performance Computing Workshop, 2003.

[22] Zhongmei Yao and Ben Choi, "Bidirectional Hierarchical Clustering for Web Mining," Proc. of the 2003 IEEE/WIC International Conference on Web Intelligence, pp. 620-624, 2003.