Distributed Object Space Cluster 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

In this paper we propose a new cluster architecture for parallel and distributed computing called Agent Space Architecture. Our architecture builds upon the notions of Agent and Object Space and utilizes Multicast Network. The building blocks for our proposed architecture consist of an active processing unit called Agent, a shared place for communication call 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. High performance of the architecture is achieved by simply adding more Agents, Spaces, or Networks. Another feature for high performance is the result of using Space as cache. This results in reducing repeated computations and response times comparing to the original concept of Object Space. Our architecture 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 Engines.

1 Introduction

Parallel and distributed computing has great potential for exploiting the vast computational power of millions of PC's 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 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.

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 Network. The building blocks for our proposed architecture consist of an active processing unit called Agent, a shared place for communication call Space, and a communication medium called Multicast Network. Multiple building blocks are organized to form a parallel and distributed computing architecture.

1.1 Organization

This paper is organized into five sections. The next section reviews works related to this research. Section 3 provides details of our proposed Agent Space Architecture. Section 4 highlights benefits resulting from our proposed architecture. And, Section 5 provides conclusion and future research.

2 Related Works

In this section we briefly review the concept of Object Space and its related architectures. Then, we highlight existence highly parallel systems for search engine applications.

2.1 Object Space and Related Architectures

Our proposed architecture is built upon an extended notion of Object Space [5]. The notion of Object Space is to have a pool of objects communicating through a shared medium. The Object Space is the shared medium that simply acts as a rendezvous for different processes or objects that meet there either to serve (worker) or be served (master) without the

This research was supported in part by Center for Entrepreneurship and Information Technology (CEnIT), Louisiana Tech University, Grant iCSe 200123.

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

BenChoi.info

knowledge of each others identity, location, or specialization. The communication between the objects is loosely coupled, which dismantles the rigid client-server, master-slave pattern, or tightly coupled interaction. Object Space offers three key advantages. (1) It offers load-balancing. Load balancing is worker driven. As long as there is work to be done and workers are available to do work, the workers will continue to perform the work. (2) It is scalable. Since objects are relatively independent and as longs as there are a sufficient number of objects, adding workers improves performance. And (3), it is robust and fault tolerance. Since there are several workers serving the space, failure of a worker will not bring the whole system down.

Other variations of Object Space are JavaSpace [5], IBM's TSpaces [10], TONIC [11], JINI [15], and TupleSpace [12]. The original initiators of TupleSpace, David Gelernter and Nicholas Carriero, argued that using Tuple Space we can create computational model for everything [16]. They suggest that any model based on Tuple Space is highly versatile whether it means for communication or coordination.

Several architectures based on the notion of Object Space have been proposed. One 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. These agents are supposed to play different roles based on a Role Timer [8] that is any role is constrained with certain duration and once that duration times out, the agent is free to play another role. The roles are maintained by a separate Role and Routing Agent which is responsible for writing the roles into the space.

A framework for cluster computing using JavaSpace [5], Object Space for Java, has been described in [9]. The framework, other than having a generic master and worker module, has a network management module. The network module is responsible for monitoring the state of the workers and uses the state information to schedule tasks to the workers. Load distribution for workers is done based on the state of their CPU utilization.

JavaSpaces has also been used for scientific computation [13]. It is used for image processing, computing the value of Pi, and Monte-Carlo Particle Shielding. It supports the perspective that JavaSpace solution has merits when used in high performance computing. The technologies like JavaSpace offer cheap solutions for high performance computing particularly when one is concerned with cost, development, deployment, and administrative efforts involved in building a high performance distributed system [13].

2.2 Search Engine Cluster Architectures

Search Engines such as Google and Yahoo represent a challenging application for parallel processing. They handle millions of requests in a single day from millions of users. The architectures of these search engines require high performance, high scalability, high availability, and fault tolerant. It is challenging to develop an architecture that meets all the requirements of search engines.

Google Architecture

Google search engine architecture [2-4] has a centralized control and is clustered with vast server farm of 15000 homogeneous PC's sharing the work load. Each of the PC has 256MB to 1GB of RAM, two 22GB or 40GB disks, and run Linux operation system. The PC's are connected with 100Mbit Ethernet to a gigabit Ethernet backbone [3]. The architecture lets different queries run on different processors. Individual queries utilize multiprocessors due to the partitioning of the overall index of the database. Most of the servers are serving up some fraction of the index. The index is partitioned into individual segments, and queries are routed to the appropriate server based on which segment is likely to hold the answer. Google employs fault tolerant software [4] to monitor the activities of individual servers and use public libraries like MPI's to express parallel processing.

Inktomi Architecture for Yahoo and MSN

The search engine architecture [1] of Inktomi Corporation [7] serves portals such as Yahoo, HotBot, Microsoft, Geocities, NTT "goo" Tokyo, and many more. It is a cluster based architecture utilizing RAID arrays with special focus on high availability, scalability and cost-effectiveness [6].

Queries in the Inktomi architecture are dynamically partitioned across multiple clusters or workers. The huge database is distributed and each segment specializes in handling certain set of sub queries. A query first arrives at the manager which then selects a single worker from a set of workers and the query is redirected to the selected worker. The worker sends the queries to all nodes that are tightly coupled with it through Myrinet [14]. All nodes run same software and handle HTTP. Each node is specialized and is

linked with a certain subset of the massive distributed database. Each node works on its portion of the database and sends back the response. Responses are collected by the designated worker. Worker sends the response back to the client. Furthermore, global load balancing is done to distribute the traffic across the heterogeneous cluster.

Inktomi architecture as explained in [1] has been designed with a mission critical attitude. Each and every failure is logged and analyzed. Weekly reporting of availability metrics is also done.

The "consistency" property that is preserving the consistency of the data and the "isolated" property that is concurrent transactions should behave as if each were the only transaction running in the system are sacrificed by the Inktomi architecture in order to achieve high availability. That is, the whole architecture has been designed to be highly available at the cost of consistency in the result set. Inktomi, further ensures high availability by carrying periodic investigation of the essential infrastructure [1].

As an effort to ensure Fault Tolerance, Inktomi sacrifices a part of the distributed database when a node linked to it is lost. That is, just pretend that the portion of the database is temporarily unavailable.

The "Smart Client" [1] of Inktomi architecture have been customized to let the queries get distributed via load balancing in an environment where interaction between any pair of node is tightly coupled.

The explicit regulation of the queries to their respective destinations through tightly coupled interaction and maintenance of mission critical attitude without doubt is able to achieve high availability and fault tolerance but deployment and administration of such a system still remains cumbersome and dependent upon a manager that redirects the queries to the worker nodes, where a lost of a node connected to a crucial segment of a database might render the result set incomplete.

AltaVista, Lycos, and Excite Architecture

AltaVista, Lycos, and Excite make use of large SMP supercomputers [1] and as such fault tolerance is good only with multiple replicated SMP's. Use of large SMP allows fast access to a large memory space. Database is stored and processed on one machine. Processors handle queries independently on shared database. All these factors lead to fixed limited scalability, expensive maintenance and costly replications.

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 call 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.

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 store 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. 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

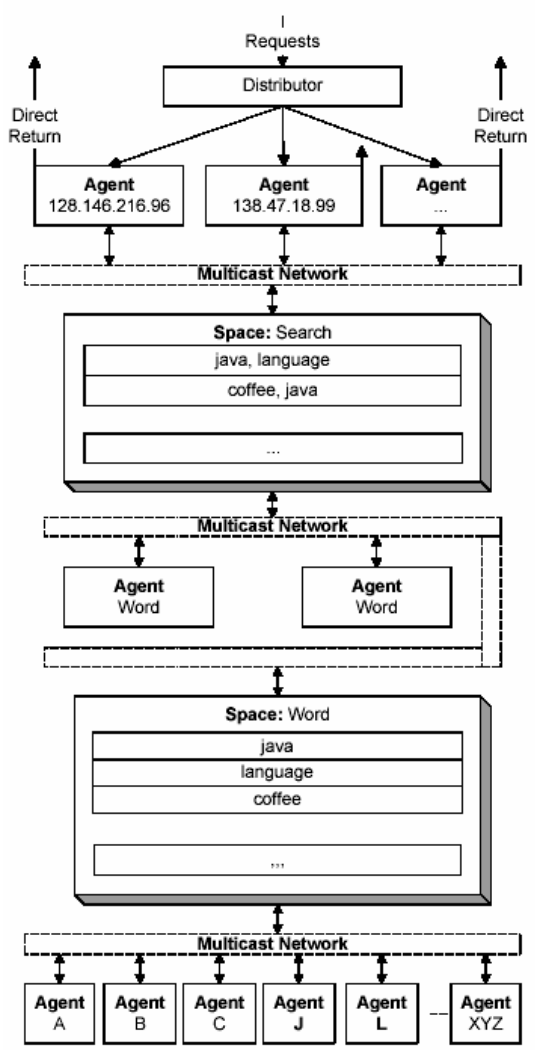

Figure 1. Agent Space Architecture for Search Engines

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 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.

3.5 Design for Search Engine Application

The basic idea of our proposed architecture for search engine applications is illustrated by using two search examples (see Figure 1). A user types "java language" and submits the search request through a client computer that has an IP address 128.146.216.96. The request is routed by a distributor to a specific agent. The agent is identified by the IP address. If no such an agent exists, an agent will be assigned to handle the request from the client computer. Agent 128.146.216.96 parsed the request. It sends a search task for "java, language", through a multicast network, to a shared medium called Space. In this case, a space is used for storing search tasks. It is identified as Space Search. Meanwhile, another user types "coffee java" and submits the search request through a client computer 138.47.18.99. The distributor routes the request to Agent 138.47.18.99 which puts the search task "coffee, java" into the Space Search.

As shown in Figure 1, there are two agents called Agent Word for handling words. These agents communicated, through a multicast network, to Space Search and Space Word. One of the agent takes the search task "java, language" from the Space Search. It puts the tasks "java" and "language" into the Space Word, which store tasks for search words. Meanwhile, another agent takes the search task "coffee, java". It puts task "coffee" into the Space Word. While trying to put task "java" into the Space Word, it finds that the task is already there. In this case, it does not need to put the task again.

As shown in the figure, there are many agents identified by characters, such as Agent A, B, and C. These agents communicate, through a multicast network, to the Space Word. They actively look for tasks to be handled. Agent A will handle search for words starting with character 'a'. While looking for task in Space word, Agent J finds that it can handle word "java". It performs a search of "java" on its database that contains data of web pages having words starting with character 'j'. It returns the search results into Space Word. Meanwhile, Agent L will handle word "language" and Agent C for "coffee". They return their search results into the Space Word.

The Agent Word who is waiting for the search results of "java" and "language" will then pickup the search results from Space Word. It will perform an intersection of the results of "java" and "language". It

then returns the results of the intersection into Space Search. Similarly, The Agent Word waiting for the search results of "java" and "coffee" will return its results of intersection into Space Search.

Agent 128.146.216.96 checking the Space Search finds that the search results for "java, language" are ready. It reads the results and generates a HTML page. It finally sends the resulting HTML page directly to computer 128.146.216.96. Meanwhile, Agent 138.47.18.99 reads the results and generates its HTML page for returning directly to its client computer. Finally, the users in those computers will see their search results.

4 Benefits of Our Proposed Architecture

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.

5 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 initial experiences with the architecture indicate that the system is easily configurable, extensible and hence mitigates the management issues confronted by existing search engine architectures.

Currently we are building a computing system based on our proposed Agent Space Architecture. The system will be used to run our Information Classification and Search Engine [19]. The proposed architecture is general enough for other high demand applications that require parallel and distributed computing.

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] http://www.Cs.nyu.edu/ courses/ spring98/ G22.3033.10/ lectures/lect0414.pdf

[7] 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] 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.