Distributed Hash Table Based Design of Soft System Buses

Mohammad Reza Selim[1], Takumi Endo[1], Yuichi Goto[1], and Jingde Cheng[1]
[1]Department of Information and Computer Sciences
Saitama University, Saitama, 338-8570, Japan
{selim, endo, gotoh, cheng}@aise.ics.saitama-u.ac.jp

ABSTRACT

Large scale computing systems which can run for unlimited time and can evolve over time have a growing demand in this modern information society. Soft System Buses (SSBs) were proposed to provide middleware platform support to such computing systems. This paper proposes a design of SSBs based on Chord – a distributed hash table protocol for large scale peer to peer systems. Our design fulfills the requirements of SSBs like scalability, automatic recovery from failures, data preservation and incremental runtime upgradeability and maintainability. An assessment of our design from the view point of the requirements of SSBs is also presented.

Categories and Subject Descriptors

C.2.4 Distributed Systems – *Distributed applications*; C.2.1 Network Architecture and Design – *Distribute networks*

General Terms

Design, Reliability

Keywords

Middleware, Soft System Bus, Distributed Hash Table, Chord Protocol

1. INTRODUCTION

Modern information society is more and more dependent on the large scale distributed systems. In many cases continuous and persistent functioning of such systems is essential. Motivated by the needs to build such computing systems, Cheng proposed a design methodology, called Soft System Bus (SSB) based methodology [2,3], to build large scale systems that functions continuously all the time without stopping its interactions even when it is being maintained, upgraded or reconfigured, it has some trouble, or it is being attacked. A system built using this methodology is called a Soft System Bus Based System (SSBBS). An SSB is a middleware of such systems [12]. However, till now no one designed such a middleware which could satisfy its requirements simultaneously.

A Distributed Hash Table (DHT) is a scalable data structure to build large-scale Peer to Peer (P2P) based distributed applications [11]. A DHT based P2P system is inherently scalable, recoverable from failure and decentralized in nature [11]. Among many candidates of structured P2P lookup protocols, Chord is the most matured one and suitable to build large-scale and scalable DHT based systems [4, 5, 13].

Conference name: Infoscale 2007, June 6-8, 2007, Suzhou, China
Copyright number (LaTeX \crdata{}): 978-1-59593-757-5

To build SSBs, in this paper, we propose a Chord based design which is suitable from the view point of scalability, data preservation, automatic recovery from failures, unified interface and incremental runtime upgradeability and maintainability.

The next section presents a brief description of SSBs and requirements of them. Chord based design of SSBs is presented in section 3. Section 4 evaluates the design hypothetically from the view point of the requirements. Section 5 presents how our designed SSB differs with other works. Finally, concluding remarks is presented in section 6.

2. SOFT SYSTEM BUSES

An SSBBS (Fig. 1) consists of a number of components and one or more SSBs [3]. The components are connected to the SSBs. They convey data/instructions from one component to another, provide unified interface to the components and preserve data/instructions if the destination component is not connected to the SSB. An SSB consists of one or more nodes called Data/Instruction Stations (DISs) which collectively provides the functionalities of an SSB.

There are two types of components in an SSBBS: one or more general purpose permanent Control Components (CCs) and some application specific Functional Components (FCs) [3]. Any two components must use the unified interface of the SSBs to interact and no direct interaction among the component is allowed.

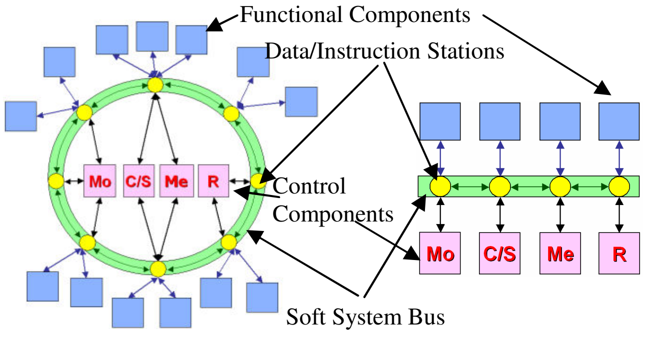

Fig 1. Circular (left) and linear (right) SSB based systems

Bellow we summarize the requirements of SSBs. A detailed analysis can be found in [12].

R1. Scalability: For running unlimited time, SSBs must support scalability of itself as well as the SSBBSs.

R2. Data Preservation: They must provide this facility so that if a component is not present, the data/instructions destined to that component can be stored in corresponding DIS.

R3. Availability and Reliability: At least one (arbitrary) DIS must be running all the time. The states of a DIS must be recoverable in case of failure.

R4. Dynamic Connectivity: Since a DIS may fail, SSBs must allow a component to select and connect to its currently associated DIS dynamically.

R5. Runtime Upgrade and Maintenance: There must have a way to upgrade and maintain the SSBs at runtime.

R6. Runtime Adaptability/Re-configurability: This requires because SSBs will evolve over time and they may operate in dynamic environments.

R7. Transmission Model: The components may need to communicate in a many to many style.

R8. Unified Interface: SSB must provide it to the components.

R9. Reconfigurable Security: SSBs must ensure it so that security mechanism can be adjusted with the evolution of SSBBSs.

3. DHT BASED DESIGN

3.1 Chord Based Approach

Chord lookup protocol [13] provides good performance in its operation. With *N* nodes in a Chord ring, each node maintains information about only $O(log_2N)$ other nodes, and a lookup only needs $O(log_2N)$ messages. A new node only increases the *N* in the $O(log_2N)$ lookup time. This means that the Chord is highly scalable.

If a node fails and produces a gap in the Chord ring, it may cause incorrect lookup. To increase robustness, each node maintains a successor list of size *r*. Therefore, the Chord ring is robust and failsafe, and the degree of reliability is tunable.

At runtime, a node can be removed from or added to the ring requiring relatively little movements of keys [13]. Therefore, the ring can be upgraded and maintained efficiently at runtime.

Therefore, use of Chord to design SSBs fulfills the requirements of scalability, availability and reliability and runtime upgrade and maintenance of SBBs (but not of components). To fulfill other requirements we have added two layers above the Chord layer.

The first one is called the Replication Layer which provides data preservation facility and dynamic connectivity. It also improves reliability and fault tolerance and provides incremental upgrade and maintenance of components. The second one which is called Component Interaction Layer fulfills unified interface as well as point to point and point to multi point transmission requirements. The other two requirements R6 and R9 have not been considered in our current research.

In our approach, each component and DIS has a unique ID (e.g., IP address, name) to identify one from another. A consistent hash function is used to convert the component ID into an *m*-bit key (note that unlike Chord we call hashed ID of a component a key). Similarly, same hash function assigns an *m*-bit unique hashed ID to each of the DISs.

Like Chord, the keys and DIS hashed IDs are treated as points and they are ordered on a circle modulo 2^m. Fig. 2 shows a circular keyspace (with *m=5*) used to implement an SSB. The circular keyspace is split into contiguous segments whose endpoints are the DIS hashed IDs. If i_1 and i_2 are two adjacent hashed IDs, then the node with hashed ID i_2 owns all the keys that are greater than i_1 and less than or equal to i_2. This ownership of keys represents which component is tapped to which DIS in the SSB. If a component *c* has a key *k*, the component will be tapped to the first DIS whose hashed ID is equal to or follows *k* in the ordered circle. Like Chord, we call this DIS the successor of *k*.

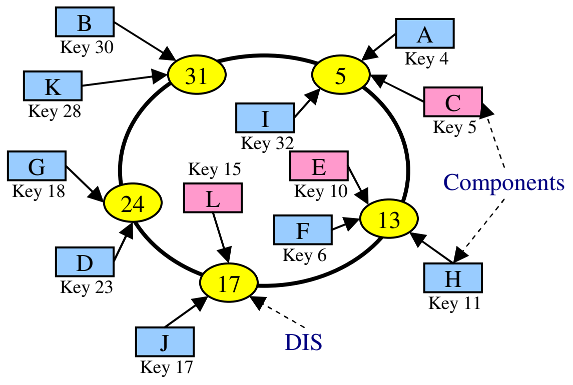

Fig. 2. Chord based SSB in a circular keyspace with m=5.

3.2 SSB Architecture

We organize the functionalities of a DIS into three different layers as shown in Fig. 3. Each layer may have some vertical interfaces through which a layer communicates with its upper or lower layer or the components (in case of the top most layer) and horizontal interfaces through which it communicates with a peer layer in a another DIS. The vertical interfaces within a DIS are implemented by function calls, while horizontal interfaces by some RPC/RMI.

3.2.1 Routing Layer

The bottom layer of a DIS is the Routing Layer. This layer has functionalities similar to the Chord Layer used in Cooperative File System [4, 5] or Key Based Routing Layer described in [6]. Its two principal functions are to keep the finger table up to date in case of joins, failures or departures of DISs and to find the successor of a component or a DIS.

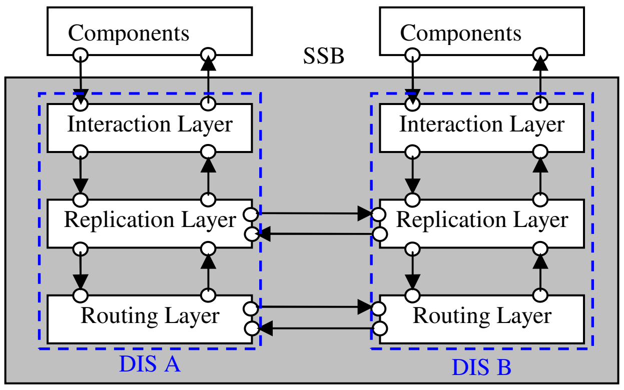

Fig. 3: Architecture of an SSB and interaction with components

3.2.2 Replication Layer

This layer has three main functions which are described bellow.

I) Managing Redundant Components

We assume that an SSBBS may contain redundant copies of critical and/or frequently used stateless components. Each copy has a different ID from one another. The purpose of redundant copies of a component is three fold: a) fault tolerance, i.e., if one copy of a component fails, the others will serve resulting in minimum effect on the system, b) load balancing, i.e., the load of frequently used components will be distributed over the redundant copies, and c) improving routing performance, i.e., the message is most likely to travel to the nearest (in the overlay) copy of a component.

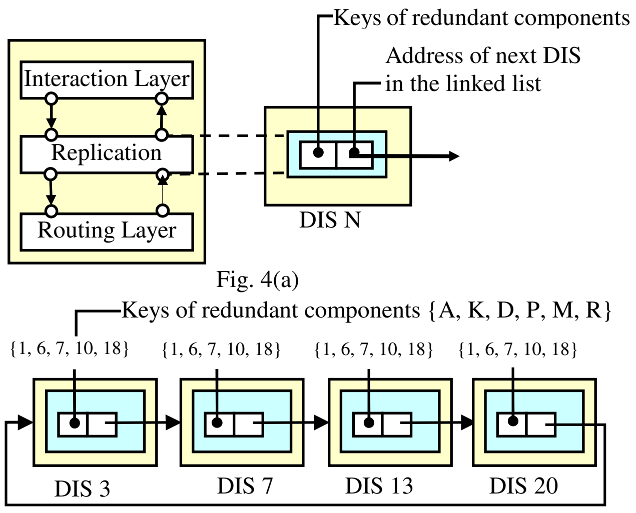

Fig. 4(b)

Fig. 4: (a) Structure of a node of the linked list (b) The linked list of the redundant components' DISs.

The Replication Layer creates a circular linked list of the redundant/replica components' DISs. In Fig. 4 each of the components of *A, K, D, P, M* with keys *1, 6, 7, 10, 18* respectively is a copy of another and they are connected to the DISs *3, 7, 7, 13, 20* respectively. Then these DISs form a circular one way linked list. The linked list is sorted according to the hashed IDs of the DISs. Each node in the linked list contains the set of the keys of the components participating in the redundancy.

To improve load balancing and routing efficiency, we use the concept of *redundant component list cache*. When a component sends a data/instruction to another component, as part of the message, its DIS sends the redundant key set of the sending component. Next time the destination DIS can reroute a data/instruction to the nearest redundant component. In this way requests to a component can be rerouted by a DIS to the nearest component in the overlay resulting in improvement of load balancing as well as routing efficiency. A cache is valid only for a time duration which we call *cache validity time*.

II) Replicating

Replication is made by a DIS to *r* number of successors. Two types of data are replicated: the data/instructions that can not be delivered to the components because of the unavailability of the components due to failure or deliberate departure for upgrade, maintenance or other reasons and current session information, i.e., information related to the connected components including the redundant component lists.

III) Forwarding Data/Instructions

After receiving a data/instruction from the Component Interaction Layer, the Replication Layer applies the hash function on the address of the destination component and produces the key. It then calls the interface of the Routing Layer to find out the IP address of the successor DIS of the key. The Replication Layer then sends the data/instruction to this DIS.

3.2.3 Component Interaction Layer

The data/instructions received by this layer from the components are handed over to the Replication Layer to send to the destination components. After getting a data/instruction from the lower layer, it is delivered to the components by this layer also.

Multicast communication is obtained by grouping the components in this layer and enforcing the components of a group to tap to one arbitrary DIS. We achieve the grouping of components by simply appending an additional field in the ID of a component to designate the group. Obviously, the hash function now must be applied on the group ID field. If one component does not want to be included in a group, it can simply repeat its own ID in the group field.

4. DISCUSSION

In this section, we check, against each of the requirements, if our design is able to fulfill or not.

R1 is satisfied. An SSB implemented using our design is scalable because adding a new DIS only increases *N* in *O(logN)* routing table size and lookup time. Cost of adding, removing or recovering a DIS is not very high due to consistent hashing [13].

As the number of components increases the Component Interaction Layer needs to keep track of more number of sessions. However, for a typical DIS, how many sessions or how much communication load or what frequency of arrival and departure of components are tolerable, can only be known after evaluating our design experimentally or with simulator.

R2 is satisfied. We have achieved the data preservation facility by storing and replicating data/instructions in the destination DIS.

R3 is partially satisfied. An SSBS will fail, if no DIS is running. Current technology can not ensure that a machine runs all the time. The Chord ring will fail only if *r* number of consecutive DISs fails simultaneously. But probability of such failure is very low for a reasonable value of *r*.

R4 is satisfied. Even if a connected DIS fails, the running components connected to it will not be disconnected, rather using the replicated session information the components will be reconnected to the new responsible DIS automatically. But at the first time, in order to connect to an SSB, like any other P2P system, a new component or DIS must know the address of aq running DIS.

R5 is satisfied. We have considered incremental runtime upgrade and maintenance where all the DISs or components can be upgraded or maintained one or more at a time but not at all. Since a component or a DIS can easily be removed from or added to an SSB without hampering the services, it provides a good environment for incremental upgrade and maintenance. However, in such upgrade and maintenance mechanism, as both the old and new versions will run in the same system, they must be compatible.

R6 is not satisfied. Our design itself does not provide sufficient facilities to automatically adapt to changes in the environment or the system itself. This is one of our future works.

R7 is satisfied. Many to many communication primitive is adopted using group based communication. A component can send messages to all the members of a group by only one request. However, if a component wants to send messages to a subset of the group or outside the group, multiple messages must be sent by the component. These cases are inefficient but rare in an SSBS.

R8 is satisfied. The Component Interaction Layer has some APIs which are used by all the components in order to interact. In this way an SSB provides unified interface.

R9 is not satisfied. We have not considered the security issues in this paper because we think that ensuring security would be premature at the current stage of our research.

5. RELATED WORKS

SSBs are neither publish/subscribe (e.g., Heremes[11], TIBCO[14], Siena[1]) nor RPC/RMI (e.g., Java RMI, CORBA) based middlewares. Unlike publish/subscribe paradigm, which does not fit in many types of systems, each component must know to which component it is going to send a message. Unlike RPC/RMI based middleware, which does not scale well, SSBs are asynchronous. Message Queuing middlewares like IBM MQSeries [8] and Oracle Advanced Queuing usually integrate some form of publish/subscribe-like interaction. These server oriented middlewares do not scale well to large populations of consumers because of the use of traditional IP based (or similar) networks other than overlays and the additional interactions need to maintain transactional, timing, and ordering guarantees.

SSBs require providing supports to the systems having a point to point or point to multi-point communication model in a paradigm where communicating components must know each other. To our knowledge, this is the first attempt to use P2P overlay network in such a paradigm. Traditional way of building such distributed systems directly over IP like networks are not naturally scalable.

Chord and other structured P2P based DHT is mainly used in cooperative file systems (e.g., CFS [5], PAST [7]). In such systems files or part of files (fragmented) are mapped to the overlay nodes based on the key produced from the files or attributes of the files. Other DHT based systems uses almost similar approach by storing values in the nodes selected based on keys. A DHT based approach usually implements a simple store and retrieve functionality. Unlike traditional DHT based approach, we are using the overlay network as a channel to communicate among components but not as a storage. Although in an SSB the data/instructions are stored temporarily, they are not retrieved or searched like in a cooperative file system. In addition, we have included a new functionality in the layer above the Chord Layer – managing the redundant components.

6. CONCLUDING REMARKS

DHT based Chord protocol has been an attractive substrate for distributed applications for its simplicity, provable correctness and good performance. Chord has already been used in many applications [5]. In this paper we use Chord to design SSBs - middlewares for an scaleable, long lived, highly reliable, runtime upgradeable and maintainable system called an SSBBS.

However, our design has not been evaluated experimentally yet. Currently we are in the middle of building a simulation based on OverSim [10] in Omnetpp [9] simulation environment. This simulation will evaluate our design from a number of perspectives including performance, load balancing, scalability and robustness.

7. REFERENCES

[1] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC 2000), 2000.

[2] J. Cheng. Comparing Persistent Computing with Autonomic Computing. In Proc. of 11th IEEE-CS Int. Conf. on Parallel and Distributed Systems. Vol. II, pp. 428-432, 2005.

[3] J. Cheng. Persistent Computing Systems as Continuously Available, Reliable, and Secure Systems. In Proc. of 1st IEEE-CS Int. Conf. on Availability, Reliability and Security, pp. 631-638, 2006.

[4] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger. Building peer-to-peer systems with Chord, a distributed lookup service, In Proc. of 8th Workshop on Hot Topics in OS, 2001.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In Proc. of the 18th ACM Symposium on OS Principles, 2001.

[6] F. Dabek, B. Y. Zhao, P. Druschel, J. Kubiatowicz and I. Stoica, Toward a Common API for Structured Peer-to-Peer Overlays. In Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003.

[7] P. Druschel and A. Rowstron, PAST: A large-scale, persistent peer-to-peer storage utility, HotOS VIII, Schoss Elmau, Germany, 2000.

[8] R. Lewis. Advanced Messaging Applications with MSMQ and MQSeries. QUE, 1999.

[9] OMNet++: Discrete Event Simulation System. http://www.omnetpp.org/

[10] OverSim: A Flexible Overlay Network Simulation Framework. http://www.oversim.org/

[11] P. R. Pietzuch. Hermes: A Scalable Event-Based Middleware. Technical Report, Computer Laboratory, University of Cambridge, 2004.

[12] M. R. Selim, T. Endo, Y. Goto, and J. Cheng. A Comparative Study between Soft System Bus and Traditional Middlewares, in R. Meersman, Z. Tari, P. Herrero et al. (Eds.), "On the Move to Meaningful Internet Systems and Ubiquitous Computing: OTM 2006 Workshops, Montpellier, France, Oct. 30 – Nov. 3, 2006", LNCS 4278, pp. 1264-1273, Springer-Verlag, 2006.

[13] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. Kaashoek, F. Dabek, H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking, Vol. 11, Issue 1, pp. 17 – 32, 2003.

[14] TIBCO. TIB/Rendezvous (White Paper), 1999.