
©2003 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
An Architecture of Internet Based Data Processing Based on Multicast and Anycast Protocols

Shui Yu, Wanlei Zhou
School of Information Technology, Deakin University, 221 Burwood HWY, Burwood, VIC 3125, Australia
{syu,wanlei,mlan}@deakin.edu.au

Abstract: Most of the current web-based application systems suffer from poor performance and costly heterogeneous accessing. Distributed or replicated strategies can alleviate the problem in some degree, but there are still some problems of the distributed or replicated model, such as data synchronization, load balance, and so on. In this paper, we propose a novel architecture for Internet-based data processing system based on multicast and anycast protocols. The proposed architecture breaks the functionalities of existing data processing system, in particular, the database functionality, into several agents. These agents communicate with each other using multicast and anycast mechanisms. We show that the proposed architecture provides better scalability, robustness, automatic load balance, and performance than the current distributed architecture of Internet-based data processing.

Keywords: anycast, algorithm, architecture, Database

I. Introduction

The Internet has become a popular platform for commercial applications, such as E-commerce, remote education, online conference, and so on. Among the applications, data processing is a critical component, for example, banking is a typical data processing application. With the exponential and ever increasing number of Internet users, it is necessary that Internet applications provide effective, accurate and reliable services.

Current Internet based data processing systems, e.g. web-based databases, usually use the three-layer architecture. The disadvantages of the model are obvious: 1. Current web-based systems are a concentrated system essentially, the requests are done in limited servers. 2. Heterogeneous platforms problem in the Internet. 3. Management problems.

One of the characteristics of the current Internet is that it does not have a total controlled management mechanism: every administrator configures his own domain using his own style. This is its advantage as well. The feature brings a difficulty of management for the Internet based distributed systems.

Performance is always a hot topic in computer applications and plenty of researches have been done in this area. It is happening in the Internet based research too. [2] surveyed the art in locally distributed web-server systems, all the methodologies contribute something for performance, but from a higher level, we find that the architectures are concentrated essentially.

As we noticed that the workloads of the Internet are not balanced, workloads of some parts are very heavy, and at the same time, that of other parts maybe very light. The Internet does not have a very effective method of workload balance in the global range.

In our opinion, the essential problem of the Internet based distributed system is that the architecture is a concentrated model, or it is not distributed enough. It is still based on a few servers to process the incoming requests, most of the jobs are processed in one server, therefore it is ineluctable that a kind of network will generate when more requests are forwarded to one server, because the computing capability and resource of one server is limited.

Base on the previous analysis, in order to get rid of the inherent disadvantages of the current architecture of the Internet based distributed systems, we propose an Internet based distributed system architecture. In the architecture, not only the system is distributed, but also the functionalities of an original server are distributed on the Internet. The proposed architecture decomposes the functionalities of an existing data processing system, e.g. a database, into a number of agents. All agents communicate with each other using multicast and anycast mechanisms.

The rest of the paper is organized as follows. Section 2 discusses the related work about multicast in database and anycast. In section 3, we present a novel architecture of distributed Internet based data processing. The algorithms applied for the proposed model are described in section 4. Finally, in section 5, we summarise the paper.

II. Related Work

A. Related Research of Multicast in Database

Distributed replication provides high availability, fault-tolerance and enhanced performance. But we must pay for these benefits: replication adds great complexity to the system development [5] [13]. Most of all, replication jeopardises data consistency. In turn, mechanisms have to be employed to enforce the data consistency. Maintaining the data consistency is very expensive.

Multicast [3] is defined as a service which tries to send a packet to every member of a multicast group. Its capability has been recognized as an important facility for networks and the Internet because of its growing usage in distributed systems. [7] presented four protocols, broadcast all protocol, broadcast writes protocol, delayed broadcast writes protocol and single broadcast transactions protocol, for distributed replicated databases that take advantage of atomic broadcast systems to simplify message passing and conflict resolution in hopes of making replication efficient.

These protocols can be applied to replicated database recovery as well [8]. [11] proposed a family of replication...
protocols based on multicast in order to address some of the concerns expressed by database designers regarding existing replication solutions. All these work show that multicast service is a good solution for the data synchronization and data recovery for distributed systems.

B. Related Research on Anycast

The original work by Partridge, Mendez, and Milliken [15] proposed the idea of anycast for the IP next generation, and discussed its network layer support. They defined IP anycasting as: A service provides a stateless best effort delivery of an anycast datagram to at least one host, and preferably only one host, which serves the anycast address. The idea of anycast meets the requirements of mirrored or replicated servers in the Internet, therefore a number of research is quickly conducted in the area. Anycasting research, as defined by the original authors, began in network layer, and researchers have archived some results [9][10][17] in network layer.

At the middle of 1990s, some researchers found the limitations of network-layer anycast, for example, inflexibility and limited supported by current routers, hence, they presented the idea of application-layer anycast [1], [4], focusing the research on anycast in the application layer. The application-layer anycast is compatible with the nature of current Internet facilities and suites for current application requirements too.

The architecture of application-layer anycasting is shown in Figure 1. A client tries to find a service from the replicated servers on the Internet. First of all, the client sends an anycast query to an anycast resolver to decide which server among the replicated servers is the "best". Then an anycast response is obtained, which consists of the "best" service server's website name or its IP address. The rest of the transaction is a traditional unicast operation.

The combining of multicast and anycast service is natural, and the cooperation can provide better service for the Internet applications [9]. The combining of anycast and multicast offers a bi-directional service for the Internet based distributed data processing systems: multicast takes the responsibility of data synchronization among the multicast group, and anycast takes the role for finding the "best" server in the anycast group, furthermore, anycast is a good methodology for server load balance and network load balance as well.

III. Architecture of the Internet Based Data Processing Model

As we have mentioned, the essential problem of the Internet based distributed system is that the architecture is a concentrated model, or it is not distributed enough. It is still based on a few or a number of servers to process the incoming requests. In order to get rid of the inherent disadvantages of the current architecture of distributed systems, we must break through the limitation of the original architecture.

As we know, database is used widely for data processing. It guarantees data integration, data recovery, etc. In our model, we broke the functionalities of a database into several relative agents, and distribute the agents all over the Internet. Currently, we define seven kinds of agents for Internet based data processing: interface agent, case agent, data engine agent, log agent, resolver agent, space agent, and synchronization agent, as shown in Figure 2.

![Figure 1. Architecture of application-layer Anycasting](image)

Some anycasting routing algorithms [1] [12] [16] [17] [18] have been proposed. [12] takes use of round trip time on an anycast router for server selection decision for network-layer anycasting. [1] proposed a network status and server load mixed application-layer anycasting algorithm, but the data of anycast resolver is updated periodically based on periodically probing on network performance and server load. [18] presents a requirement-based probing algorithm for application-layer anycasting. The algorithm issues probing packets to all the replicated server for each anycast query. The theory analysis shows that the requirement-based probing algorithm is better than the periodical probing algorithm.

For the convenient of explanation, we give three definitions for the proposed architecture bellow.

**Definition 1**: Data processing Node. If a set of agents can complete a given transaction, then we call that set as a data processing node logically. The agents may be distributed anywhere in the Internet. In the remaining of this paper, we use node as short for data processing node.

The components of a node are dynamic, and request driven. A node may include part of the seven types of agents as mentioned earlier, or all of them. The resource employed by the node will be released immediately after the transaction's completion. In some circumstances, two or more nodes may share an agent if the agent is chosen by them, as shown in Figure 3.
The number of nodes in the system is not a constant, but all the storage space should be mirrored. All the replicated nodes are shared by all the Internet clients, each client choses the "best" node for himself/herself, and all of the nodes should provide the same service, therefore, the data consistency between the replicated node is very important, and the replicated nodes should be synchronized as soon as possible. For a request, it will be processed in one node, and then the transaction will be executed in the other nodes to guarantee the global data consistency. We give two definitions about the nodes here.

**Definition 2: Original Node.** A node, which processes a user's requests and the requests are not processed by any other node in the system, is called an original node.

**Definition 3: Replica Node.** A node, which processes a user's requests and the requests have been processed at least once in the system by another node, is called a replica node.

Once there comes a request, an original node will be created using the anycast protocol, therefore all the agents of the node is the "best" ones for the node from their categories, respectively, and the transaction will be completed by the collaboration of the agents, and then all the replica nodes will be synchronised through the multicast mechanism.

The functionalities of these agents are defined as following.

- Interface agent takes the responsibility of receiving user requests, and then notifies a local Case agent to process the transaction. Once the result is achieved, the interface agent will assemble the result and deliver the final result to users.
- Case agent processes a given transaction. First of all, the case agent figures out which agent(s) is/are required for the case, then finds the "best" agent(s) for this transaction using the anycast mechanism of the resolver agent. In the end it dispatches the deputies to the agent(s) respectively.
- Data engine agent acts as a light weight database engine. It interprets the incoming requests, and executes the commands.
- Log agent provides log service. Log service includes log space management and log service, which guarantees atomicity of transactions.
- Space agent takes care of space management for data deposit.
- Synchronisation agent multicasts the executed transactions in local node to the replica nodes in the group.
- Resolver agent takes care of anycast and multicast related information and services. The resolver agent maintains two groups, namely, anycast group and multicast group, which includes the anycast group information and multicast group information respectively.

**IV. Algorithms for Internet Based Data Processing Model**

In this section, we will describe the algorithms for the proposed model. From the view point of a system, there are three algorithms: algorithm of distributed data processing; algorithm of anycasting; and algorithm of atomic multicast update.

**A. Algorithms for Distributed Data Processing**

Generally speaking, a distributed system tries to synchronise among the mirrored servers periodically, hence during the interval of synchronization the data in the mirrored servers are not consistent. Therefore the whole system can not share the non-consistent servers. During the synchronization, the system uses a lot of its computing capability to update the non-integrated data among the mirrored servers, for this reason, the performance of new requests are badly affected. As a result, it is a dilemma for choosing the interval of system synchronization.

The algorithm of the proposed architecture is described below.

1. When a request comes to a case agent, the case agent tries to figure out which agents are needed to complete the request.
2. The information about wanted agents are delivered to a local resolver agent.
3. The resolver agent feedback back the "best" agents respectively.
4. The request will be executed among the agents in parallel.
5. Flush log of transaction in log space
6. Flush transaction result to storage spaces
7. Purge the log in log space
8. Once the request is completed, the executed scripts will be delivered to the other nodes in the system using multicast methodology of the resolver agent. Then the data in the whole system are synchronised.
9. At the same time, the result is transported to the interface agent and then delivered back to the user.

The algorithm is listed in list 1.

**Strategy 1:** A queue for log stores all the logs for transactions, once a predefined percentage, say 80%, of the queue is full, then flush data to storage spaces.

**Strategy 2:** Once a transaction is completed, flush the log to the storage space.

The strategy 1 has a better performance and flexibility, whereas the second one has a better reliability.
At the original node,

```java
interface agent {
  case agent (request_id) // transaction is transported
  return result // to user
}
```

```java
case agent (request_id) {
  while (original transaction == true) {
    // check the resource for the transaction
    if (read deposit space) then space_agent (case_id, size)
    if (read log) then log_agent (case_id)
    dataAccessAgent (case_id)
    return result // to interface agent
  }
  release resource // release the occupied resource for this transaction
}
```

```java
space_agent { // space management
  call resolver_agent
  return space resource if the "best" space resource
}
```

```java
log_agent {
  call resolver_agent
  return log resource if the "best" log resource
  logProcess() // construct log processing
}
```

`DataEngine_agent { // data processing
  call resolver_agent
  locate DataEngine_agent // the "best" DataEngine resource
  Data engine process
  logFlush() // flush the log data into hard disks
  dataFlush() // flush the data process result into hard disks
  purgeLog() // purge the log information in the log area
}
```

```java
synchronization_agent { // synchronize the replicated servers in the group
  multiagent transactions
}
```

Listing 1 Algorithm for Distributed Internet-based Data Processing Model

For data consistency, we need forward operations based on log once a crash happens. For convenience, we use strategy 1 as an example to explain our transaction forward algorithm. If a server crashes, there are three possibilities for log and transaction states, listed in Table 1.

<table>
<thead>
<tr>
<th>State</th>
<th>Log</th>
<th>Transaction</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>not flushed</td>
<td>not flushed</td>
<td>no need</td>
</tr>
<tr>
<td>2</td>
<td>flushed</td>
<td>not flushed</td>
<td>need</td>
</tr>
<tr>
<td>3</td>
<td>flushed</td>
<td>flushed</td>
<td>no need</td>
</tr>
</tbody>
</table>

Table 1 states of log and transaction

From Table 1, we find that when the state 2 happens, it is necessary to forward transactions to guarantee data consistency. Moreover, when transactions are flushed, the log area will be cleared immediately. Therefore, the log area is empty or not is the only prerequisite for transaction forward or not, the algorithm is listed in list 2.

In order to compare the performance of the common web transaction and anycast based web transaction, we conducted a simulation using network simulator 2 [14].

```java
if (log is not empty) {
  read log(s)
  execute log(s)
  purgeLog()
}
```

Listing 2 Algorithm of Transaction Forward

The scenario is that there are two LANs connected by a connection. All the servers are located in one LAN, and all the clients are located in another LAN. There are simulated web traffic as background. We tried to download a file from two replicated servers using common method and anycast method, respectively. The size of the file varies from 5Mb to 50Mb, and the result is shown as in Figure 4. The vertical axis is the values of the method deduct the anycast method.

![Figure 4: Performance Difference between Common and Anycast Web Transactions](image)

From Figure 4, we found that anycast method is faster than the common method in most of the individual download transactions. The difference of the first and fourth points comes from the unstable background web traffic.

In order to compare the performance of common data replication and multicast data replication, we conducted another simulation. The scenario is the same as the last simulation. The size of data be transported is 2.5Mb. The multicast algorithm is Dense Mode (DM in short) [14].

**Definition 4:** Replication Transaction Time (RTT, in short). To a group of replicated servers \(S_1, S_2, \ldots, S_i, \ldots, S_n\), the time lasted for each replication is denoted as \(T_i\), then the replication transaction time for a replication is 

\[
\max \{T_i, i = 1, 2, \ldots, n\}.
\]
We vary the bandwidths among the servers from 1.5Mb, 2Mb, 5Mb, 10Mb, to 50Mb for each simulation. The result is shown in Figure 5. The vertical axis is the values of common RTT, deduct the multicast RTT.

From Figure 5, we find that from the viewpoint of RTT, the multicast replication is better than the common replication, especially when the network bandwidth is limited.

In another simulation, we vary the number of replicated servers and exam the RTT of multicast replication and common replication respectively. For the multicast replication, we use two multicast algorithms: Centralized Multicast (CM in short) [14] and Dense Mode (DM in short). The result is shown in Figure 6.

![Replication Transaction Time](image)

**Figure 6**: Comparison of Multicast and Common Replication with Difference Replicas.

From figure 6, we find that when there are only 2 or 3 replicas, the common replication algorithm is better, but with the increasing number of replicas, the multicast replication algorithms are better than the former.

**B. Algorithms of Application-Layer Anycast**

The algorithm for anycasting is an important component in the proposed architecture. In this paper, we only discuss the application-layer anycasting algorithms. There are two kinds of application-layer anycasting algorithms, which will be presented below.

**B.1 Periodical Probing Algorithm**

The critical problem of application-layer anycasting is how to map an anycast query into one or more IP addresses. [1] presents 4 metrics about how anycasting performs: 1) server response time, 2) server-to-user throughput, 3) server load, and 4) processor load. As we found that in [1], the foundation of anycast resolver algorithms is the remote server performance probing, which based on periodical probing, we call it as periodical probing algorithm (PPA in short). [1] mixed the different methods together in practical applications. There are several disadvantages for periodical probing, such as, accuracy problem, network load problem, completeness problems, resolver server load problem, and so on [19].

We presented an algorithm, called requirement-based probing algorithm (RPA in short) [18], which can overcome most of the disadvantages of the periodical probing algorithm. The main idea of the requirement-based probing algorithm is described below.

When an anycast query is received by an anycast resolver, the resolver will send probing packets, such as ping, to each member in the anycast group, respectively. In this case, the probed servers will respond for the ping requirements, respectively. If a server's workload is heavy or performance is bad, then the responding must last longer than a server whose load is light or performance is good. Therefore the probing packets can not only probe the servers' load or performance at that short period, but also probe the network load at the same period. Based on the analysis, we consider that the first responsive server is the best one among the anycast service group, because the responsive time represents the network performance and server performance, then the anycast resolver will submit the IP address of the best server to the client via the anycast response. The client then tries to find the best server using the traditional IPv4 procedures.

We simulated the two algorithms in the scenario of downloading data from anycast servers. We present the performance of the two algorithms here. For the periodical probing algorithm, there is a resolver system data update process. During the system data update, the performance of the resolver is dramatically decreased, and we assume that it can not provide resolver service during that period. In the simulation, we configure that the resolver update interval as 100 seconds, and the updating period varies at 1%, 10%, and 15% of the interval, respectively. We check the downloaded bytes for the requests every 100 seconds.

In Figure 6, RPA represents the algorithm of requirement-based probing algorithm, and 0.01, 0.10, and 0.15 represent periodical probing algorithm with the system update time is 1%, 10%, and 15% of system update interval respectively.

![Bytes (MB)](image)

**Figure 6**: System Service Performance Comparison of RPA and PPA.

From Figure 6 we learn that at each checkpoint the RPA transfers more data than the PPAs, that means at the viewpoint of system service performance, RPA is better than PPA.
B.3 Algorithm of Atomic Multicast Update

A transaction is a sequence of read and write operations on the data items that is executed atomically. For atomic multicast, in this paper, we consider it satisfies the following properties [6]:

1. If a node broadcasts a message \( m \), the primitive ensures that the message will be delivered to all operational nodes.
2. If a node delivers message \( m \), then all operational nodes deliver \( m \).
3. If nodes \( p \) and \( q \) deliver broadcast messages \( m \) and \( m' \), then \( m \) and \( m' \) are delivered in the same order at all nodes.

Multicast mechanism used in database replication is researched by numbers of papers [7] [8] [11], we embed the idea into our model. The detail of atomic multicast update algorithm for our model is described below.

1. A transaction is initialised at one node.
2. Case agent passes the statements to the relative local agent, at the same time, the case agent copies all the statements except the commit statement, and multicasts the statements to the multicast group using resolver agent.
3. The replica nodes execute transactions.
4. For both of the original node and the replica nodes, a transaction, \( T_i \), tries to execute a read locally, if the information is not available locally, then the resolver agent will submit the read operation to the rest of the members using anycast mechanism to get the related information quickly.
5. Once the original transaction is committed, the resolver agent will issue the commit demand to all the members in the rest of the replicated group.
6. After that, all the members in the multicast group are synchronized.

The algorithm can reduce the possibility of transaction deadlock, because the algorithm takes us of all the readable resource among the mirrored data processing nodes. Besides, the algorithm can decrease the possibility of long transactions, because it is not necessary to wait for unlock of local resource.

V. Summary and Future Work

In this paper, we propose a distributed Internet based data processing model. The unique characteristics of the proposed architecture are as follows:

- Essentially, the proposed architecture is more distributed than the current distributed architectures. The functionality of an original server is decomposed into smaller functionalities as agents. All the agents are distributed in the Internet, and they can compose a data processing node flexibly.
- From the viewpoint of theory, the performance of the proposed architecture is improved comparing with the current architectures, because the new model decreases the possibility of system bottleneck and the possibility of transaction deadlock.
- The management of the new model is even simpler than that of existing model, because the usage of anycast and multicast mechanisms.

We introduced three algorithms for the proposed architecture: algorithm of distributed data processing describes the data processing in a logical node, algorithm of atomic multicast update deals with the data replication among the distributed nodes, and the algorithm of application-layer anycasting servers used for information searching.

We conducted some simulations for the algorithms, and the result shows that the performance of our proposed model is better than the current models. We will try to implement a prototype of the proposed model, and apply the model to practical applications.

As an architecture of data processing, there are a number of details; in this paper we only explored the skeleton of the proposed architecture. There are some other aspects, such as security, authentication, etc, are needed and worthy to be researched.

References