P2P 네트워크상에서 MapReduce 기법 활용

(An Application of MapReduce Technique over Peer-to-Peer Network)

임 건 길 † 이 재 기 ††
(Jian-Ji Ren) (Jae-Kee Lee)

요 약 본 논문의 목적은 P2P 네트워크 상에서 동적 환경 애플리케이션을 지원하기 위한 MapReduce의 설계이다. MapReduce는 클라우드컴퓨팅 중에서 대용량 데이터의 병렬처리를 위해서 개발된 소프트웨어 프레임워크이다. P2P 기반 네트워크의 특징은 노드 고장이 언제든지 발생할 수 있으며, 이런 노드 고장을 제어하기 위해 Pastry라는 DHT 라우팅 프로토콜의 사용에 초점을 맞추었다. 본 논문의 결과는 프레임워크가 양호한 계산 효율과 확장성을 유지하는 가운데 P2P 네트워크 시스템의 다양한 애플리케이션에 적용될 수 있음을 보이고 있다. 향후 몇 년 동안은 P2P 네트워크와 병렬 컴퓨팅이 산업과 학계에서 매우 중요한 연구 및 개발 주제로 자리 잡을 것으로 확신한다.

키워드 : MapReduce, 병렬 컴퓨팅, DHT, P2P 네트워크

Abstract The objective of this paper describes the design of MapReduce over Peer-to-Peer network for dynamic environments applications. MapReduce is a software framework used for Cloud Computing which processing large data sets in a highly-parallel way. Based on the Peer-to-Peer network character which node failures will happen anytime, we focus on using a DHT routing protocol which named Pastry to handle the problem of node failures. Our results are very promising and indicate that the framework could have a wide application in P2P network systems while maintaining good computational efficiency and scalability. We believe that, P2P networks and parallel computing emerge as very hot research and development topics in industry and academia for many years to come.

Key words : MapReduce, parallel computing, DHT, P2P Network

1. 서 론

The advances in Internet, access technologies, availability of high bandwidth and parallel programming have opened new opportunities to dominate distributed computing research. In recent years, Peer-to-Peer (P2P) architecture has aroused much attention both in research communication and industries. It has played a vital role in wide spreading growth of computing applications including grid computing, utility computing and other distributed computing applications. P2P networking is promising for its several favorable characteristics, such as self-organization, self-configuration, self-healing, easy maintenance, high scalability and reliable service.

Cluster Systems and P2P Systems are the two most common types of resource sharing systems currently in wide use. These two systems evolved from different communities and serve different needs. Cluster systems interconnect computer clusters, storage systems, instruments, and in general the available infrastructure of large scientific computing centers in order to make possible the sharing of existing resources, such as CPU time, storage, equipment, data, and software applications. Most Cluster systems are of moderate-size, they are centrally or hierarchically administered and there are strict rules governing the availability of the participating resources.

Challenges in scaling computing to increasingly-large datasets have become a serious issue. The trends on data growth and processor speed improvement suggest that some form of parallelization is required to process data. It is clear that datasets readily available today and the types of analyses that researches wish to conduct have outgrown the capabilities of individual computers. The only practical recourse is to distribute the computation

· 본 논문은 2008학년도 동아대학교 학술연구비 지원에 의하여 연구되었음
· 이 논문은 2008 학술심포지움에서 'P2P 네트워크상에서 MapReduce 활용'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 동아대학교 컴퓨터공학과
jimey@donga.ac.kr
†† 종신회원 : 동아대학교 컴퓨터공학과 교수
jklee@dau.ac.kr

논문접수 : 2009년 4월 14일
심사완료 : 2009년 6월 30일

across multiple cores, processors, or machines. The traditional cluster computing could not handle the problem that dealing with large-scale data-intensive at a low cost. The leading example is Google. Recently, Jeffrey Dean and Sanjay Ghemawat in Google Corporation proposed a parallel framework, MapReduce [1], which can provide this necessary simplicity, while at the same time offering load balancing and fault tolerance to process 20 petabytes of data per a day [2].

There has been existed some MapReduce architectures, such as Google's [1] and Hadoop [3]. They are based on a master-slave model. A job is submitted by a node as a master that selects idle nodes as slave and assigns each one a map or reduce task. When all map and reduce tasks have been completed, the result was returned to the master node. The failure of a slave node is managed by re-executing its task on other idle slave nodes, while data structures can be written out so if a failure occurs a new copy can be restarted from the last save. The master failures are not considered by current MapReduce implementations. As designers say that it is assumed that a master failure is unlikely so if it fails the user must restart his/her job.

In the P2P environments, contrary to cluster system, node failures will happen anytime, like grid computing systems and volunteer computing systems, where nodes join and leave the network at an unpredictable rate. Therefore, providing effective mechanisms to manage the nodes failures fundamental to exploit the MapReduce framework in the implementation of data-intensive applications or other network computing applications in those P2P environments where current MapReduce mechanisms could be unreliable. The goal of our work is to design a new mechanism that taking the MapReduce framework over the P2P network.

In this paper, we discuss the mechanisms that using MapReduce framework over DHT P2P network to compute. In this P2P network, each node can be selected as master or slave by the mechanism and search each other by Distributed Hash Table. The role assigned to a given node depends on the current characteristics of that node, and so it can change dynamically over time. Thus, at each time, a limited set of nodes is assigned the master nodes, while the other nodes are assigned the slave nodes or idle nodes. Moreover, a main master has some backup master nodes, which will manage it as usual in MapReduce. The master nodes will periodically check the status of the job on its backup master nodes using checkpoint. In case those backup master nodes detect the failure of the using master node, they will elect a new master node from themselves and restart the job from the last available checkpoint.

This paper is organized as follows: in section 2, we give details description of the problem of using MapReduce over P2P network and our solutions. Section 3 introduces the simulation studies that use the Parstry protocol to simulate the data transmission over P2P network with MapReduce process. Finally, we conclude this paper in section 4.

2. MapReduce over P2P Network

2.1 Overview

As noted, our system is a distributed infrastructure for MapReduce to share data and computability over the low-cost P2P network. The whole architecture of MapReduce over P2P network includes four basic roles, shown in Fig. 1: User (U), tracker server (T), master nodes (M), slave nodes (S) and idle nodes (I). As mentioned above, nodes in P2P network are dynamically assigned the master nodes role, slave nodes role, idle nodes role or tracker server; hence each node could change their characters over time. The architecture of node in our system includes three lays: Access Service Application, Distributed File Locating, and P2P Routing Protocol, shown in Fig. 2.

The MapReduce over P2P network consists of participating nodes running the MapReduce application services. These nodes in P2P network communicate using a DHT P2P routing algorithm. The P2P system automatically manages the joining and leaving of nodes, locates files and services, and runs the services requested by users.

2.2 P2P Routing Protocol

The P2P routing protocol in our system is responsible for maintaining the organization of the cooperation nodes and it is also responsible for

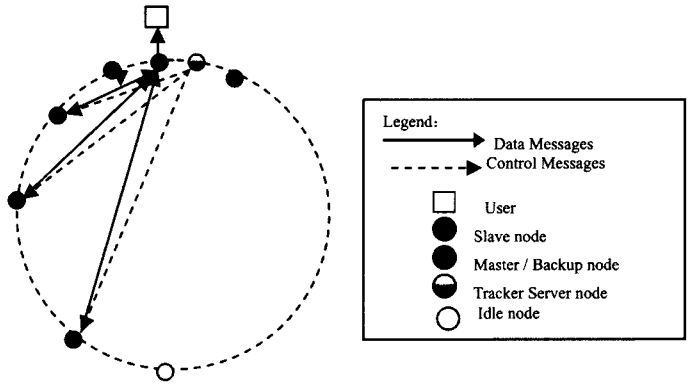


Fig. 1 The architecture of MapReduce over P2P

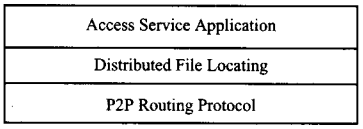


Fig. 2 Node architecture

locating files in the P2P network. The current routing protocol used in our system is Pastry [4], a P2P routing protocol based on a distributed hash table (DHT). Each peer in Pastry is assigned a 128-bit peer identifier (NodeID). The NodeID is used to give a peer's position in a circular NodeID space, which ranges from 0 to $2^{128}-1$. The NodeID is assigned randomly when a peer joins the system and it is assumed to be generated such that the resulting set of NodeIDs is uniformly distributed in the 128-bit space. A Pastry node efficiently routes the message to the node with a nodeId that is numerically closest to the key, among all currently live Pastry nodes. The expected number of forwarding steps in the Pastry overlay network is O(log N), while the size of the routing table maintained in each Pastry node is only O(log N) in size (where N is the number of live Pastry nodes in the overlay network).

On another hand, each Pastry node keeps track of its immediate neighbors in the nodeId space (called the leaf set), and notifies applications of new node arrivals, node failures and node recoveries within the leaf set. The last one, Pastry takes into account locality (proximity) in the underlying Internet; it seeks to minimize the distance messages travel, according to a scalar proximity metric like the ping delay. Pastry is completely decentralized, scalable, and self-organizing; it automatically adapts to the arrival, departure and failure of nodes. It allows for replication of data and finds the closest replica.

2.3 Failure Recover Protocol

In the following we describe, through an example, how a master node failure is handled in our P2P architecture. We assume the starting situation where U is the user that submits a MapReduce job; M nodes are the masters and S nodes are the slaves.

The following steps are performed to submit the job and recover from a master node failure (see Fig. 3. All queries are supported by DHT.):

1) U sends a request to tracker server to get the list of the available master nodes, each one characterized by a workload index that measures how busy the node is. Tracker server orders the list by ascending values of workload index, takes the first element as primary master node and the next k elements as backup master nodes. In this example, M1 is chosen as primary master node and M2 and M3 as backup master node (k = 2).
2) U submits the MapReduce job to M1 along with the names of its backup nodes M2 and M3. M1 notifies M2 and M3 that they will act as backup nodes for the current job. This implies that M1 will periodically backup the entire job state (e.g., the assignments of tasks to slave nodes, the locations of intermediate results, etc.) on M2 and M3. Beside tracker server will periodically check whether M1 is alive.

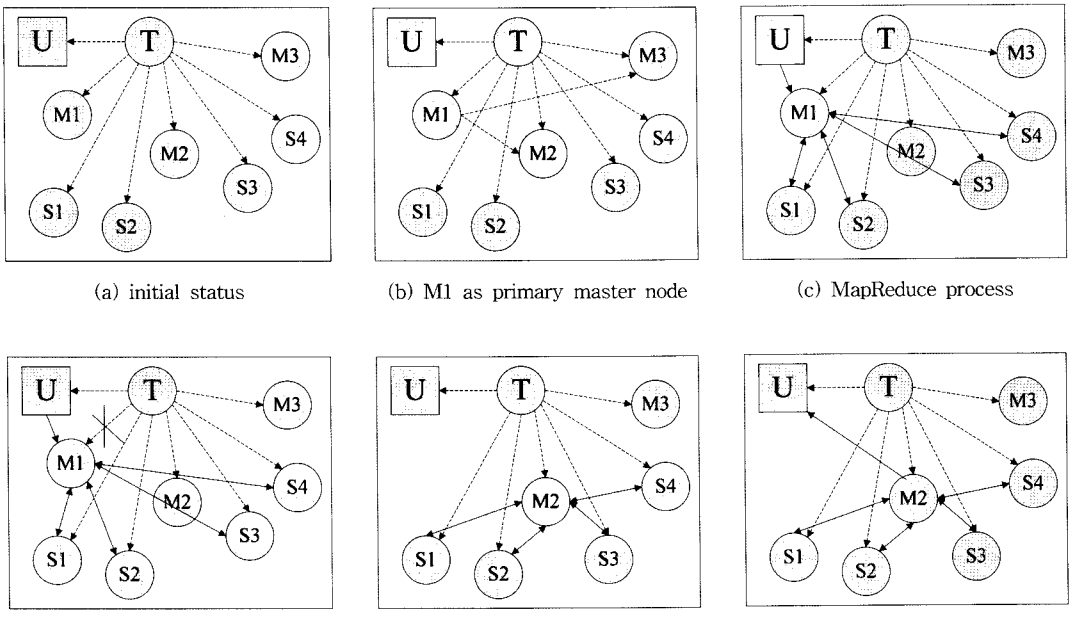
3) M1 queries tracker server to get the list of the available slave nodes, choosing (part of) them to execute a map or a reduce task. As for the masters, the choice of the slave nodes to use can be done on the basis of a workload index and other performance metrics. In this example, nodes S1, S2, S3 and S4 are selected as slave nodes. The tasks are started on the slave nodes and managed as usual in MapReduce.
4) The primary master M1 fails. Tracker server detects the failure of M1 and starts a procedure based on P2P routing protocol to elect among backup nodes of M2 or M3 to be as the new primary master.
5) The new primary master M2 is elected by choosing the backup node by the tracker server. The remaining k = 1 backup master nodes (only M3, in this example) continue to play the backup master node role. Then, the MapReduce job starts from the latest checkpoint available on M2.
6) As soon as the MapReduce job is completed M2 returns the result to U.

As above mention, the slave nodes can use such protocol to handle the slave nodes failures recover.

3. Evaluation and Experimental Results

In order to test our MapReduce over P2P network service performance, we build up a simulation environment. The underlying network topology used in our experiments is generated by GT-ITM [5] in the Transit-Stub fashion. They are distributed in 2 transit networks with 20 stub domains. We assign link latencies of 150ms for intertransits domain links, 100ms for intra-transit domain links and 50ms for stub-transit links and 20ms for intra- transit links. We assume that the nodes leave system rate is conformed to a Poisson distribution with parameter λ and the tracker server never leave system. The file size before MapReduce is F and MapReduced file size is f, where F = 1800MB. The file which should be processed is stored in some nodes as master nodes. The Pastry's replica ability is k, where k = 3. The MapReduce processing capability is t/MB = 20s/MB. In our simulation, we use a normalized workload model.

The result for job completion time on the total number of nodes is shown in Fig. 4. We plot the inverse of time in minutes (which shows the amount of work completed per unit time) v.s. the number of nodes used.



(a) initial status (b) M1 as primary master node (c) MapReduce process

(d) M1 failure (e) New primary master node M2 (f) MapReduce job completion

Fig. 3 Steps performed to submit a job

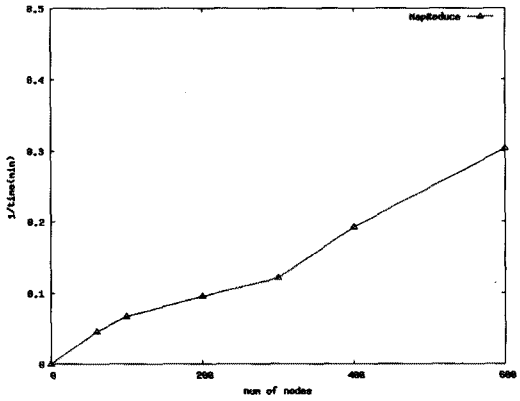


Fig. 4 Job completion time on the total number of nodes

4. Conclusion

For several years now, commodity cluster computing and P2P computing have been quite common. We have analyzed the basic differences between cluster and P2P system. MapReduce provides a restrictive programming environment which allows for a variety of tasks to be easily computed in parallel. It allows for simple parallel iteration over data and parallel aggregation of results. This paper design a MapReduce over Peer-to-Peer network architecture which is based on Pastry failure recovers protocol for dynamic environments applications. We believe that, there is much scope for further research in this area and using Peer-to-Peer architecture to implement MapReduce will be a very issue.

References

[1] J.Dean and S.Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In *Proceedings of OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pp.137-150, Dec. 2004.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol.51, no.1, pp.107-113, Jan. 2008.

[3] Hadoop http://hadoop.apache.org/core/.

[4] Pastry http://freepastry.org/.

[5] K. Calvert, M. Doar and E. Zegura, "Modeling Internet Topology," *IEEE Communication Magazine*, vol.35, no.6, pp.160-163, Jun. 1997.