Multiple Agents for Data Processing

Ichiro Satoh

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
ichiro@nii.ac.jp

Abstract. This paper proposes a distributed processing framework inspired from data processing. It unique among other data processing for large-scale data, so-called bigdata, because it can locally process data maintained in distributed nodes, including sensor or database nodes with non-powerful computing capabilities connected through low-bandwidth networks. It uses mobile agent technology as a mechanism to distribute and execute data processing tasks to distributed nodes and aggregate their results. The paper outlines the architecture of the framework and evaluates its basic performance.

1 Introduction

MapReduce is a model for processing large data sets. It was originally studied by Google [2] and inspired by the *map* and *reduce* functions commonly used in parallel LISP or functional programming paradigms. Data processing should be provided close to the sources of data, e.g., sensors and database, as much as possible to reduce network traffic. However, existing projects assume that MapReduce is executed in data centers or in-house high performance computing systems, which may be far from the sources of data. This paper proposes a novel implementation of MapReduce to process data at the edges of networks. In fact, it can satisfy the following requirements, which existing MapReduce implementations cannot support.

- The goal of our MapReduce implementation is to directly execute MapReduce processing on ambient computing systems and sensor networks, where each node may have non-powerful processor with the small amount of memory, rather than data centers and high performance server clusters.
- Networks in such systems may be low-band, often disconnected, and dynamic, e.g., wireless sensor networks. Therefore, our implementation should be available in such networks.
- There may be no database or file systems in the target systems. Instead the data that need to be processed are generated or locally maintained in the local storage of nodes.

S. Omatu et al. (Eds.): *Distrib. Computing & Artificial Intelligence,* AISC 217, pp. 147–154.
DOI: 10.1007/978-3-319-00551-5_18

- Every node may be able to support management and/or data processing tasks, but may not initially have any codes for its tasks.

The basic idea behind our implementation is to deploy data processing tasks at the nodes that have the target data at the edge of networks and aggregate the results, rather than to transmit data to servers at the center. It introduces mobile agent technology, where mobile agents are autonomous programs that can travel from computer to computer in a network, at times and to places of their own choosing. The state of the running program is saved, by being transmitted to the destination. The program is resumed at the destination continuing its processing with the saved state. Our implementation of MapReduce defines the management system and data processing tasks as mobile agents and *map* and *reduce* processing in MapReduce are provided by migrating workers, which are implemented as mobile agents, with the results of their processing. It is constructed based on our original mobile agent platform, which is designed for data processing, in particular MapReduce processing.

2 Related Work

The tremendous opportunities to gain new and exciting value from big data are compelling for most organizations, but the challenge of managing and transforming it into insights requires new approaches. MapReduce processing has been used as one of the approaches. It originally supports *Map* and *Reduce* processes [2] The first is to divide a large scale of data into smaller sub-problems and assign them to worker nodes. Each worker node processes the smaller sub-problem. The second is to collect the answers to all the sub-problems and aggregates them as the answer to the original problem it was trying to solve. *Hadoop*, which is one of its one of the most popular implementations of MapReduce, developed and named by Yahoo!. There have been many attempts to improve Hadoop in academic or commercial projects. On the other hand, there have been a few attempts to implement MapReduce itself except for Hadoop. For example, the Phoenix system [6] and the MATE system [5] supported multicore processors with shared memory. Haloop [1] and Twister [3] were designed for MapReduce-based iterative computation. Google's MapReduce, Hadoop, and other existing MapReduce implementations assume their own distributed file systems, e.g., Google file system (GFS) and Hadoop file system (HDFS), or shared memory between processors. For example, Hadoop needs to move target data from the external storage systems to HDFS via networks before its processing. Our MapReduce system does not move data between nodes. Instead, it deploys program codes for defining processing tasks to the nodes that have the data by using the migration of agents corresponding to the tasks and execute the codes with their current local data. In the literature of sensor networks, IoT, and machine-to-machine (M2M), several academic or commercial projects have attempted to support data on the edge, e.g., sensor nodes and embedded computers. For example, Cisco's *Flog Computing* and EMC's computing intend to integrate cloud computing over the Internet and peripheral computers.

3 Mobile Agent-Based MapReduce

This section outlines our mobile agent-based MapReduce processing system and compares between our system and Hadoop, which one of the most typical implementation of MapReduce. The architecture of our MapReduce system is different from existing implementation of MapReduce, including Hadoop.

3.1 Data Processing at the Edge

The original MapReduce and its clones are unable to cope cost-effectively if at all with new dynamic data sources and multiple contexts for the large amount of data, which is generated at sensors and devices. More data are generated at the edge of networks, e.g., sensors and devices, than servers, including data centers and cloud computing infrastructure. The transmission of such data from nodes at the edge to server nodes seriously affect the performance of analyzing the data and results in congestion in networks. To solve this problem, data processing tasks are defined as mobile agents and dynamically duplicated and deployed at the nodes that have the target data. Mobile agents also can directly access data from sensors and low-level file systems at their destination nodes. Our approach assumes data at nodes to be independent of one another and can be processed without exchanging data between nodes. Finally, like MapReduce, agents running on nodes carry their results to specified nodes after their processing are done to aggregate the results.

3.2 Architecture

The original MapReduce consists of one *master* node and one or more *worker* nodes and Hadoop consists of *job tracker*, *task tracker*, *name*, and *data* nodes, where the first and third corresponds to the master node, the second and forth to data nodes in the original MapReduce. Our MapReduce system has a little different architecture from Google's MapReduce and a far from Hadoop. The system itself is a collection of three kinds of mobile agents, called *Mapper*, *Worker*, and *Reducer*, which should be deployed at appropriate nodes, called *data nodes*, according to the location of the target data. They are still mobile agents so that they can dynamically deployed at nodes according to the location of the target data and available resources to process them.

3.3 MapReduce Processing

Our system supports MapReduce processing with mobile agents. Figure 1 shows the basic mechanism of the processing.

- **Map** process: a *Mapper* agent corresponds to the master node of the original MapReduce. The agent makes copies of *Worker* agent and each of the *Worker* agents migrates to one or more data nodes, which locally have the target data. They execute their processing locally at the nodes. When there are multiple

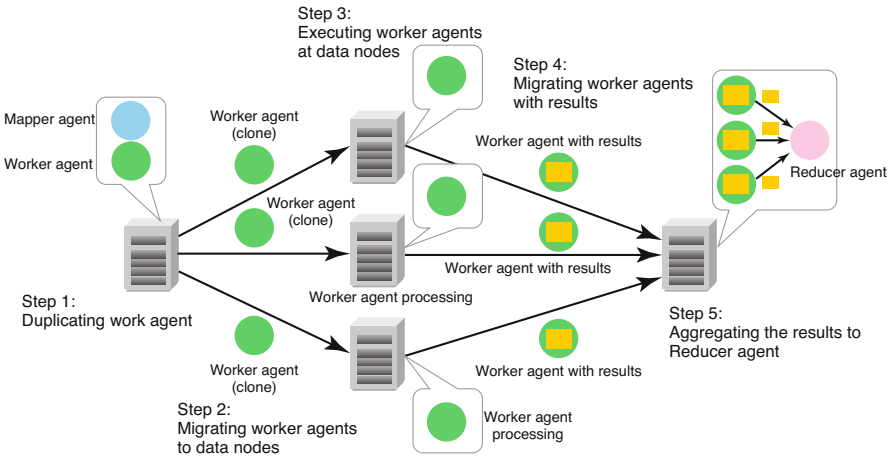

Fig. 1 Mobile agent-based MapReduce processing

Mapper agents in the same time, they can be executed in a specified schedule, e.g., sequential or parallel.

- **Reduce** process: after executing their processing, *Worker* agents migrates to the computer that the *Reducer* agent is running with their results and then send the results to the agent. *Mapper* and *Reducer* agents can be running on the same node.

Note that the amount of the results are by far smaller than the amount of target data. Each *Worker* agent assumes to be executed independently of the others.

4 Design and Implementation

This section describes our mobile agent-based MapReduce system. It consists of two layers; mobile agents and agent runtime systems. The former consists of agents corresponding to job tracker and *map* and *reduce* processing and the latter corresponds to task and data nodes. It was implemented with Java language and operated on the Java virtual machine. In our implementation, the system has been designed independently of MapReduce, because it can support other data processing approaches in the same time. As a result, our approach can be available in other existing mobile agent platforms.

4.1 Agent Runtime System

Each runtime system runs on a computer and is responsible for executing agents at the computer and migrating agents to other computers through networks. The system itself is designed independent of any data processing. Instead, agents running on it support MapReduce processing.

4.1.1 Agent Duplication

Before deploying agents at data nodes, our approach makes one or more copies of task agents. The runtime system can store the state of the agent in heap space in addition to the codes of agents into a bit-stream formed in Java's JAR file format, which can support digital signatures for authentication. The current system basically uses the Java object serialization package for marshaling agents. The package does not support the capturing of stack frames of threads. Instead, when an agent is duplicated, the runtime system issues events to it to invoke their specified methods, which should be executed before it is duplicated and it then suspends their active threads.

4.1.2 Agent Migration

Each runtime system also establishes at most one TCP connection with each of its neighboring systems in a peer-to-peer manner without any centralized management server and exchanges control messages and agents through the connection. When an agent is transferred over a network, the runtime system transmits one or more marshalled agents to the destination data nodes through TCP connections from the source node to the nodes. After arriving at the nodes, they are resumed and activated from the marshalled agents and then their specified methods are invoked to acquire resources and start their processing.

4.1.3 Agent Execution

Each agent can have one or more activities, which are implemented by using the Java thread library. Furthermore, the runtime system maintains the life-cycle of agents. When the life-cycle state of an agent is changed, the runtime system issues certain events to the agent. The system can impose specified time constraints on all method invocations between agents to avoid being blocked forever. Each agent is provided with its own Java class load, so that its namespace is independent of other agents in each runtime system. The identifier of each agent is generated from information consisting of its runtime system's host address and port number, so that each agent has a unique identifier in the whole distributed system. Therefore, even when two agents are defined from different classes whose names are the same, the runtime system disallows agents from loading other agents's classes. To prevent agents from accessing the underlying system and other agents, the runtime system can control all agents under the protection of Java's security manager.

4.2 Mobile Agent

Each agent is defined as a collection of Java objects. It is general-purposed. Instead, we provide agents with a framework for MapReduce processing. Every agent

consists of several callback methods to be invoked by the runtime system before or after the life-cycle state of the agent changes, e.g., initialization, execution, arrival, departure, suspension, and termination. It can invoke several fundamental methods used to create a new agent as its child and control the life-cycle of itself and its children, e.g., mobility, duplication, termination. To support existing data processing software for Hadoop, the current implementation can explicitly provide callback methods compatible to Hadoop's classes and interfaces, e.g., `Mapper` and `Reducer`.[1]

Unlike other existing MapReduce implementations, including Hadoop, our system does not have any file system, because nodes in sensor networks and ambient computing systems may lack enrich storage devices. Instead, it provides a tree-structured key value stores (KVSs), where each KVS maps arbitrary string value and arbitrary byte array data and is maintained inside its agent, and directory servers for KVSs in agents. To support *reduce* processing, the root KVS merge KVS of agents into itself. In the current implementation each KVS in each data processing agent is implemented as a hashtable whose keys given as pairs of arbitrary string values and values are byte array data and is carried with its agent between nodes.

5 Current Status

A prototype implementation of this framework was constructed with Sun's Java Developer Kit version 1.6 or later versions. The implementation provided graphical user interfaces to operate the mobile agents. Although the current implementation was not constructed for performance, we evaluated that of several basic operations in a distributed system where eight computers (Intel Core Duo 2 2 GHz with MacOS X 10.7 and J2SE version 6) were connected through a Giga Ethernet.

- The cost of agent duplication was measured as the left of Fig.2, where The agent was simple and consisted of basic callback methods. The cost included that of invocating two callback methods.
- The cost of migrating the same agent between two computers was measured as the right of Fig.2. The cost of agent migration included that of opening TCP-transmission, marshaling the agents, migrating the agents from their source computers to their destination computers, unmarshaling the agents, and verifying security.

We constructed word counting from texts.[2] Fig.3 shows the basic structure of our word counting by using mobile agent-based MapReduce with the screenshots of the word counting.

[1] It does not support fully compatible methods, since it does not intend to use existing programs for Hadoop.

[2] Word counting is one of the most typical examples of Hadoop.

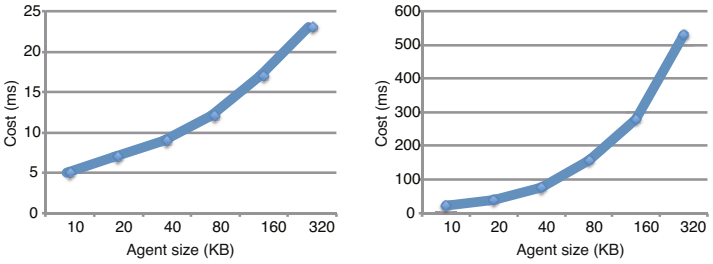

Fig. 2 Mobile agent-based MapReduce processing

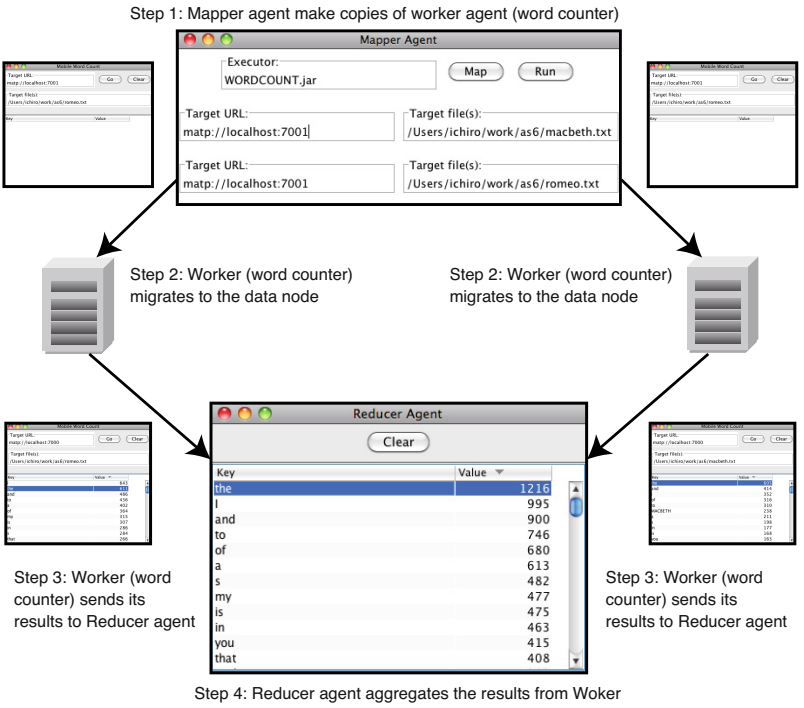

Fig. 3 Mobile agent-based MapReduce processing

6 Conclusion

We presented a novel distributed processing framework inspired from MapReduce processing. It was designed for analyzing data at the edges of networks and constructed based mobile agents. It introduces mobile agent technology so that it distributed data processing tasks to distributed nodes as a *map* process and aggregates their results by returning them to specified servers as *reduce* process.

References

1. Bu, Y., Howe, B., Balazinska, M., Ernst, M.D.: HaLoop: Efficient Iterative Data Processing on Large Clusters. Proceedings of the VLDB Endowment 3(1) (2010)
2. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th Conference on Symposium on Opearting Systems Design and Implementation, OSDI 2004 (2004)
3. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.H., Qiu, J., Fox, G.: Twister: a runtime for iterative MapReduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC 2010). ACM (2010)
4. Grossman, R., Gu, Y.: Data mining using high performance data clouds: experimental studies using sector and sphere. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2008), pp. 920–927. ACM (2008)
5. Jiang, W., Ravi, V.T., Agrawal, G.: A Map-Reduce System with an Alternate API for Multi-Core Environments. In: Proceedings of 10th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (2010)
6. Talbot, J., Yoo, R.M., Kozyrakis, C.: Phoenix++: modular MapReduce for shared-memory systems. In: Proceedings of 2nd International Workshop on MapReduce and Its Applications (MapReduce 2011). ACM Press (2011)