MAP task allocation strategy in an ARM-based Hadoop cluster by using local storage as split cache

Bongen Gu and Yoonsik Kwak*

Department of Computer Engineering,
Korea National University of Transportation,
Chungju-Si, 27469, Chungbuk, Republic of Korea
Email: bggoo@g.ut.ac.kr
Email: yskwak@ut.ac.kr
*Corresponding author

Abstract: The increase of power consumption makes the cost of cluster operation higher. One approach for reducing power consumption is to establish a cluster with small nodes which equip a low-power, high-performance processor. Since many lowpower consumed nodes do not have storage devices, a separate storage system is required to store large-volume data while nodes mount this storage space to save data. When a Hadoop cluster is configured in such a condition, each node's access to a storage results in excessive network load and delays the execution of Hadoop Map tasks. In this study, we propose a newmap task scheduling policy for Hadoop. This policy transmits multiple splits to nodes at once to reduce network load. In addition, local storage space of nodes is used as a cache for a split, which shortens the time to access splits, so this policy can reduce the execu tion time of Hadoop applications.

Keywords: Hadoop; ARM; cluster; map; scheduling.

Reference to this paper should be made as follows: Gu, B. and Kwak, Y. (2016) 'MAP task allocation strategy in an ARM-based Hadoop cluster by using local storage as split cache', *Int. J. Advanced Media and Communication*, Vol. 6, No. 1, pp.65–72.

Biographical notes: Bongen Gu received his PhD in Computer Engineering at Kyungpook University. He is currently a Professor at the Department of Computer Engineering, Korea National University of Transportation. He leads the Computer System Laboratory (CSL). His research interests include issues related to high-performance computing, parallel system, storage, embedded system, mobile system and agricultural product monitoring system.

Yoonsik Kwak received his PhD in Electronics Engineering at Kyung-Hee University. He is currently a Professor at the Department of Computer Engineering, Korea National University of Transportation. He leads the Microprocessor and Embedded System Laboratory (MESL). His research interests include issues related to embedded system, sensor network application and agricultural product monitoring system.

This paper is a revised and expanded version of a paper entitled 'TERM-based MAP task scheduling in an ARM-based Hadoop cluster' presented at *Int. Conf. on Platform Technology and Service, Platform Technology Letters*, Jeju, 26–28 January, 2015.

1 Introduction

Processor and memory technologies are advancing in terms of integration and performance as the advancement of semiconductor processing technology contribute to the refinement of elements and circuits. In the early 2000s, semiconductor technology makes manufacturer enabling to implement 100 nm- or 20 ns-scale semiconductor as described by Kim et al. (2013). High integration and high-speed operation of semiconductors, however, have caused an increase of power consumption and excessive heat generation. To prevent semiconductor elements from being damaged due to the excessive heat, additional cooling systems to decrease the temperature are necessary and inevitably require more power consumption. The excessive power consumption of processors and cooling systems is the hindrances to increase the performance and throughput of the system, which is pointed out by Kim et al. (2013), Kogge et al. (2008) and Kim and Kim (2010).

High-performance computing (HPC) is technology to provide a high level of computing power for a large-scale operation in various areas such as science, engineering, and social science. There are various approaches to HPC. One approach to implementing HPC is supercomputer equipped with a large number of vector processors. Such supercomputers store data to be processed in a vector or an arrangement format, and high performance is secured when each dataset is independent of each other, which is pointed out by Kunzman and Kale (2011). Recently, cluster-based supercomputers are widely used.

Another way to provide HPC is a general-purpose computing on GPU, which is called GPGPU. Weijun et al. (2009) pointed out that GPGPU can use the processing power of GPU, which was originally developed for graphics processing, to process the general tasks. And Kim et al. (2012, 2013) also pointed out that many recent types of research focus on designing supercomputers by means of GPGPU. NVIDIA Corp. develops and produces a GPGPU of CUDA structure. And, NVIDIA (http://www.nvidia.com/object/cuda_home_new.html) introduces its supercomputer models implemented by using GPGPU. Many-core processor technology is one of the approaches for HPC, which is pointed out by Mattson (2010). Many-core processors integrate from tens to hundreds of processing cores in one chip to increase the throughput.

A cluster is a set of computing nodes that work in unity to execute applications by networking computer systems that are independently operated. As the number of nodes in a cluster increases, the processing performance enhances. Even if nodes of different types and specifications are involved, a cluster can execute applications tasks. These features make cluster technology being widely used as a way to provide HPC. To process data by using a cluster effectively, it is necessary to manage nodes, partition and allocate application tasks. To do this, libraries such as Parallel Virtual Machine (http://www.csm.ornl.gov/pvm/), Open MPI (http://www.open-mpi.org/) and frameworks such as hadoop Hadoop (http://hadoop.apache.org/) are utilised.

Cluster technology makes data processing of conventional HPC application enabling. One of HPC application fields is bioinformatics. In bioinformatics field, alignment for a sequence of DNA or protein elements requires significant computing power, and thus many researchers studies on sequence alignment algorithms and systems. Some researchers, which are described in Hadoop BLAST (http://portal.futuregrid.org/manual/hadoop-blast), execute BLAST, a major sequence alignment algorithm, in a Hadoop cluster.

To provide higher performance, the number of nodes on a cluster can be increased. This situation results in increasing power consumption. As power consumption increases, nodes generate heat, so air-conditioning systems to decrease the temperature also consume more power. Many types of research have been conducted with great investments into node development based on low-power processors to decrease power consumption by nodes themselves. ARM-core processors developed by ARM Ltd. (http://www.arm.com) are low-power processors commonly used among mobile devices. Currently, they are used for low-power servers. A cluster that consists of nodes with a low-power processor may reduce power consumption.

Nodes that use a low-power ARM-core processor may not include a storage device such as HDD or involve a small-capacity storage device such as eMMC, flash memory, etc. Hence, a cluster that consists of such nodes requires a separate storage system, and in this case, each node shares one storage system. When multiple nodes share one data storage system, disk access requests from each node are concentrated on the storage system, which may lead to a bottleneck. For example, basic scheduling policies of Hadoop that allocate tasks based on the locality of a data block are not effective when all data needs to be stored in one storage system.

In this paper, we address necessities and problems when one shared storage system is established for a Hadoop cluster that consists of nodes with an ARM-core processor. We propose a scheduling policy to solve such problems. The scheduling policy proposed in this paper is not based on blocks, a basic scheduling unit of Hadoop for Hadoop map task allocation to each node, but based on a group of blocks that can be allocated to one node simultaneously. This scheduling policy reads a group of blocks allocated to each node in a storage at once so that multiple map tasks can be executed simultaneously.

This study includes the following sections. In Section 2, we introduce clusters that consist of nodes with low-power processors based on the ARM core and Hadoop. In addition, the necessity and problems of a shared storage systems on a cluster are addressed. In Section 3, we propose a Hadoop map task scheduling policy, and then we summarise our works. In Section 4, we summarise our works and presents further study.

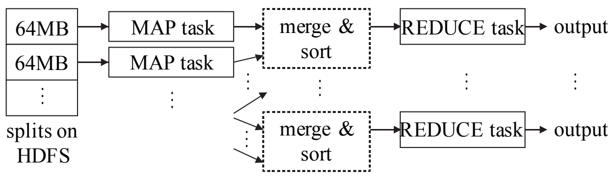
2 ARM-based Hadoop clusters

2.1 Hadoop map/reduce

Hadoop is one of the MapReduce programming frameworks, which is described by Tom (2011), appropriate for processing of large-volume data called BigData. Hadoop consists of an implementation of MapReduce model and a distributed file system called HDFS.

Figure 1 illustrates the way to process data in Hadoop. Large-volume data is partitioned to block units and stored in each node – called data node – according to HDFS policies. The basic size of one HDFS block is 64 MB, and this is called 'split' in Hadoop. Each split is basically stored in three nodes according to HDFS replication policy. Thus, each split has three copies. Hadoop scheduler allocates one split to one mapper when executing map tasks. For processing, each mapper executed in data nodes reads splits allocated to it from a disk of remote data nodes or a local disk of nodes. mappers deliver the result of map task processing to Merger. Merger merges result from each mapper and deliver them to Reducer so that it can execute reduce tasks for additional processing.

Figure 1 Data processing model on Hadoop



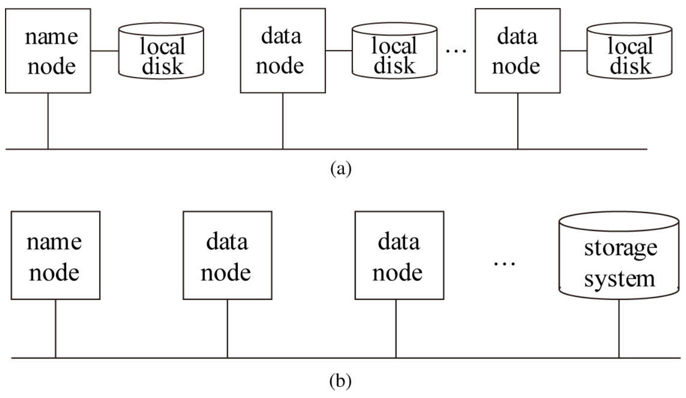
2.2 *ARM-based Hadoop clusters*

Various types of ARM-based processors developed with an ARM core are utilised to implement a low-power, light-weighted server. Some server vendors have released related products. As ARM cores continued to be developed and the performance enhanced, these cores used in designing processors for small embedded systems. Such systems support Linux as the operating system and development environments such as JDK in Java-based software development. Hence, they may be used as Hadoop cluster nodes. When Hadoop clusters are established by using ARM core nodes, cluster featured with low-cost, low-power, and high-performance is possible. Small embedded systems with an ARM core, however, may not contain a storage device such as HDD or may involve a small-capacity storage device based on a semiconductor memory such as eMMS and flash memory. Hence, when a Hadoop cluster is established by means of such embedded systems, a separate storage system is required for storing large-scale data.

Figure 2 illustrates a common configuration of Hadoop clusters and a of ARM-based Hadoop clusters. In Figure 2(a) that shows a common configuration of Hadoop clusters, each node includes a storage device such as HDD, which is used for HDFS. In other words, large-volume data stored in HDFS is partitioned into split units as previously described, and each split is stored in HDD of the data node. As mapper executed in data node *i* processes data of split *i*, the split is read by the local disk for processing. In contrast, when mapper executed in data node *j* is given split *k* for processing, this mapper receives split *k* from data node *m* which stores split *k* through the network for processing. Due to the overhead when a 64MB split is transmitted through a network, the scheduler of Hadoop uses a scheduler policy that considers locality for a split to be processed in the data node where it is located. In this way, split transmission through a network is minimised.

Figure 2(b) illustrates a Hadoop cluster configured with ARM-based data nodes. As mentioned above, each data node does not include a separate storage for large-volume data. Thus, an additional storage system is required and provides a file system mounted through NFS, Samba, etc. In this case, it is logical that the distributed file system should be formed with the storage at each data node, but the storage mounted and used by each data node is physically one storage system. Hence, when mappers executed on node *i* and *j* process splits *i* and *j*, respectively, every split is transmitted from one storage system. As a result, the network load of disk input/output of the storage system increases, and thus scheduling policies of Hadoop based on locality are actually meaningless. Just as if mapper that processes a split is allocated to a data node in which the split is stored, even if a scheduling policy that considers locality is executed by a scheduler, the split is transmitted through a network physically from one storage system. As described above, one split is stored basically in three data nodes according to HDFS replication policy. However, since these data nodes mount their file systems in one storage system, this replication policy may be unnecessary. To avoid such unnecessary duplicated splits and to consider the split store strategy based on the storage system rather than locality of splits, a new Hadoop scheduling policy is required.

Figure 2 Hadoop cluster model: (a) conventional Hadoop cluster and (b) ARM-based Hadoop cluster



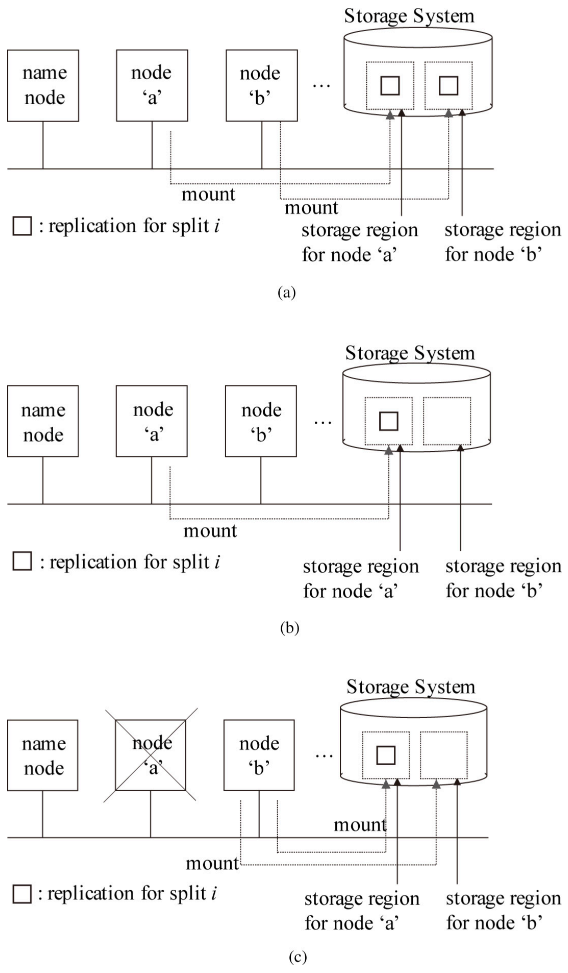
3 TERM-based MAP task scheduling

In this section, we propose a new Hadoop map task scheduling policy in consideration of the unnecessarily duplicated splits and split locality in ARM-based Hadoop cluster. Unnecessary duplication of splits that may occur when one storage system is used can be prevented by changing HDFS configuration parameter of Hadoop to avoid duplicated storage when data nodes are mounted in one storage system.

Figure 3 compares between replication-enabled and -disabled HDFS configurations. As shown in Figure 3(a), when split i is stored in data nodes a, b, and c in duplication, this split is stored in the storage areas mounted at each data node. The goal of duplicated storage is to prevent data loss due to one data node's breakdown. This goal can be achieved by securing the storage system reliability and applying a fault tolerant policy. In other words, restoration policies for a breakdown of disk need to be executed at the level of a storage system, not in the level of data nodes. Hence, HDFS replication policy is unnecessary in this case. As shown in Figure 3(b), however, when there is no need to store splits in duplication, the storage efficiency is tripled. When the access to a data node fails due to breakdown, as shown in Figure 3(c), it is possible to restore it by letting a normally operated node to mount the area because the storage space mounted by that node is still preserved in the storage even if the data node is broken down.

The map task scheduling policy proposed in this paper is not based on locality but on the number of map tasks that can be executed simultaneously on each data node. Splits allocated to each mapper are read by a storage system at once, and then the number of storage system accesses is reduced by using as cache the semiconductor memory based small-volume storage device equipped on the data node. As a result of applying this TERM-based scheduling policy, the network transmission overhead is reduced and the data access rates in splits increase.

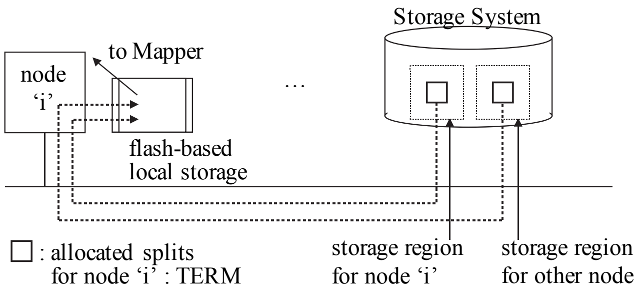
Figure 3 Fault recovery strategy on replication-disabled Hadoop configuration: (a) replication-enabled configuration; (b) replication-disabled configuration and (c) faulty case on replication-disabled configuration


Storage System
name node
node 'a'
node 'b'
...
mount
mount
□ : replication for split i
storage region for node 'a'
storage region for node 'b'
(a)
Storage System
name node
node 'a'
node 'b'
...
mount
□ : replication for split i
storage region for node 'a'
storage region for node 'b'
(b)
Storage System
name node
node 'a'
node 'b'
...
mount
mount
□ : replication for split i
storage region for node 'a'
storage region for node 'b'
(c)

'TERM' used in this paper indicates the number of splits allocated to map tasks that can be executed simultaneously in a data node. Assume that the number of map tasks that can be executed simultaneously in a data node is n, and the number of splits that the data to be processed is divided to for storage in HDFS is m. The size of TERM is n, and the number of TERM is $\lceil m/n \rceil$. The size of TERM is the minimum scheduling unit, and map tasks are allocated to each data node so that data can be processed simultaneously. Assume that the size of data to be processed in a cluster is about 6.4 GB and the number of map tasks that can be executed by the data node simultaneously is 4. The size of TERM is 4 and data consists of 26 TERMs. As TERM-based scheduling policies schedule map tasks based on TERM, 4 map tasks are allocated to each data node so that 4 splits are processed simultaneously.

A data node to which TERM is allocated reads splits to be processed by the storage system at once and stored in a storage space of data nodes. This storage works as a cache for split processing. Figure 4 illustrates interactions between the storage system and map tasks executed in the semiconductor-based local storage space of data nodes. In Figure 4, node *i* reads TERM allocated to that node at once and stores it in its own local storage space. The local storage space in a node can be eMMC or flash memory, and TERM or splits stored in that area are accessed on a read-only basis at map tasks. Thus, the average access time is reduced. Each map task processes data by means of splits stored in that area.

Figure 4 Remte/local storage for TERM-based MAP scheduling


Storage System
to Mapper
node 'i'
…
flash-based local storage
□ : allocated splits for node 'i' : TERM
storage region for node 'i'
storage region for other node

The TERM-based map task scheduling policy proposed in this paper reads splits contained in TERM at once and stores them in a local storage space of nodes that work at relatively high rates. This method removes the need for frequent access to the disk or storage system to partition or process record units of data stored in each split. As a result, network load does not increase due to frequent access to a storage, and splits stored in a high-speed local storage space are used for map task execution, which contributes to reducing the general time of map task implementation.

4 Conclusion

Among various approaches for HPC, clusters are studied in many research projects because of low cost, expandability, availability, easy maintenance, etc. A cluster is a set of computing resources that connect computer systems with independent processing capabilities so that they jointly work for one application process. As the number of nodes that join this cluster increases, the processing performance and throughput increase in proportion, but power consumption increases as well. Since more power consumption involves more costs, efforts to reduce power consumption are required.

One approach to reducing cluster power consumption is to establish a cluster by means of nodes designed with a low-power high-performance processor. An ARM-core processor was designed originally for mobile device processors, and thus, low-power consumption was one of the basic requirements. As ARM core design technology advanced, the functions and performance of ARM-core processor may be enough to execute a general application. While the cluster is implemented by using node equipped with an ARM-core processor, it is possible to establish a cluster of low-power, low-cost, and high-performance. Many of the small nodes with ARM core processors, however, do not include storage devices such as HDD for large-volume data, and thus a separate storage system may be required. In such a cluster environment, making and executing applications by means of a Hadoop framework is likely to involve frequent access to the storage system and increase of network load, which will lower the performance of map task implementation.

We described the flaw of Hadoop map task scheduling policies based on split locality on ARM-core based cluster. And we proposed a TERM-based scheduling policy as a solution. According to this scheduling policy, a node reads multiple splits from a storage system at once and stores them in a local storage space. map tasks access splits in the local storage to process data. This is to reduce network load by preventing frequent access to a storage system and utilise data cached in a local storage space for map task implementation. As a result, the general processing time is reduced.

In future, we study on the implementation of the scheduling policy proposed in this paper.

References

Kim, Y.W. and Kim, S.W. (2010) 'Technology and trends of high performance processors', *Electronics and Telecommunications Treads*, Jungsun, Vol. 25, No. 5, pp.123–136.

Kim, Y.W., Park, K. and Kim, H.Y. (2012) 'Recent treands on high performance computing system technology', *Proc. of the ITFE Summer Conf.*, pp.23–25.

Kim, Y.W., Kim, H.Y., Bae, S., Kim, H.Y., Woo, Y.C., Park, S.J. and Choi, W. (2013) 'Design fo MAHA supercomputing system for human genome analysis', *Trans. on Software and Data Engineering*, Vol. 2, No. 2, pp.81–90.

Kogge, P., Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J., Darp, S., Keckler, S., Klein, D., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, Al., Sterling, T., Stanley, R. and Yelick, K. (2008) *Exascale Computing Study: Technology Challenges in Achieving Exascale Systems*, Defense Advanced Research Projects Agency Information Processing Techniques Office, RechRep.15.

Kunzman, D.M. and Kale, L.V. (2011) 'Programming heterogeneous systems', *2011 IPDPSW*, Shanghai, pp.2061–2064.

Mattson, T. (2010) *The Future of Many Core Computing: A Tale of Two Processors*, Intel Labs Report.

Tom, W. (2011) *Hadoop: The Definitive Guide*, O'relilly.

Weijun, X., He, H., Yan, S. and Qing, Y. (2013) 'Promise of embedded system with GPU in artificial leg control: enabling time-frequency feature extraction from electromyography', *Engineering in Medicine and Biology Society*, Minneapolis, MN, pp.6926–6929.

Websites

ARM – The Architecture For The Digital World, http://www.arm.com

Hadoop BLAST, http://portal.futuregrid.org/manual/hadoop-blast

What Is Apache Hadoop, http://hadoop.apache.org/

nVidia, CUDA Parallel Computing Platform, http://www.nvidia.com/object/cuda_home_new.html

Open MPI: Open Source High Performance Computing, http://www.open-mpi.org/

PVM: Parallel Virtual Machine, http://www.csm.ornl.gov/pvm/