

RESEARCH

Open Access



A performance optimization strategy based on degree of parallelism and allocation fitness

Changtian Ying^{1,2}, Changyan Ying² and Chen Ban^{2*}

Abstract

With the emergence of big data era, most of the current performance optimization strategies are mainly used in a distributed computing framework with disks as the underlying storage. They may solve the problems in traditional disk-based distribution, but they are hard to transplant and are not well suitable for performance optimization especially for an in-memory computing framework on account of different underlying storage and computation architecture. In this paper, we first give the definition of the resource allocation model, parallelism degree model, and allocation fitness model on the basis of the theoretical analysis of Spark architecture. Second, based on the model presented, we propose a strategy embedded in the evaluation model which is easy to perform. The optimization strategy selects the worker with a lower load that satisfies requirements to assign the latter tasks, and the worker with a higher load may not be assigned tasks. The experiments consisting of four variance jobs are conducted to verify the effectiveness of the presented strategy.

Keywords: Performance optimization, Parallelism degree, Allocation fitness

1 Introduction

In recent years, big data processing framework [1, 2], especially for in-memory computing framework, enriches and develops constantly [3, 4]. The in-memory computing has appeared in our view and attracted wide attention in the industry after the SAP TechEd global conference in 2010.

With the development of the in-memory computing framework, some research results are committed to the expansion and improvement of the system. A simple and efficient parallel pipelined programming model based on BitTorrent was proposed by Napoli et al. [5]. Chowdhury et al. implemented the broadcast communication technology for the in-memory computing framework. Lamari et al. [6] put forward the standard architecture of relational analysis for big data. A study by Cho et al. [7] proposed a parallel design scheme. An algorithm using programs to analyze and locate common subexpressions was designed in a study by Kim et

al. [8]. A study by Seol et al. [9] proposed a fine granularity retention management for deep submicron DRAMs. et al. designed a unified memory manager separating the memory storage function from computing framework. In a study by Tang et al. [10], a standard engine for distributed data stream computing was designed. A high-performance SQL query system was implemented in a study by Jo et al. [11]. A parallel computing method for the applications with the differential data stream and prompt response was proposed in a study by McSherry et al. [12]. Zeng et al. designed a general model for interactive analysis. A study by Corrigan-Gibbs et al. realized the privacy information communication system of in-memory computing. A study by Sengupta et al. [13] used SIMD-based data parallelism to speed up sieving in integer-factoring algorithms. Ifeanyi et al. [14] presented a comprehensive survey fault tolerance mechanisms for the high-performance framework.

Some research results focus on the performance optimization for distributed computing framework, which may not suitable for the in-memory framework. Ananthanarayanan et al. proposed the algorithm,

* Correspondence: yingct2018@163.com

²School of Information Science and Engineering, Xinjiang University, Urumqi 830008, People's Republic of China

Full list of author information is available at the end of the article

making full use of the data access time and data locality. By analyzing the impact of task parallelism on the cache effectiveness, Ananthanarayanan et al. designed a coordinated caching algorithm that adapted to in-memory computing. By monitoring computation overhead, Babu et al. found that the parallelism of the reduce task has a great influence on the performance of MapReduce system, and the task scheduling algorithm is designed to adapt to resource status. In order to predict the response time of worker node, Zou et al. divided a task into different blocks, which can improve the efficiency of tight synchronization application. In a study by Sarma et al., the communication cost frontier model of worker node was proposed, and the tradeoff between the task parallelism and communication cost were achieved by adjusting the boundary threshold. A study by Pu et al. presented FairRide, a near-optimal, fair cache sharing to improve the performance. Chowdhury et al. proposed an algorithm to balance multi-resource fairness for correlated and elastic demands.

However, most of the current performance optimization strategies are mainly used in distributed computing framework with disks as the underlying storage, in which we pay the most attention to two aspects: task scheduling and resource allocation. Therefore, it is of practical significance to study the optimization mechanism of IMC framework from the perspective of underlying memory-based storage and computation architecture.

Therefore, we consider the degree of parallelism and allocation fitness which differs from the existing strategy. First, taking the task scheduling into consideration, the rationality of the parallelism degree of the shuffle process for the in-memory framework is easier to ignore that may directly affect the efficiency of job execution and the utilization rate of cluster resources. But the degree of parallelism is usually determined based on user experience, and it is hard to adapt to the existing state of the in-memory framework. Second, achieving the rationality of the hardware allocation, especially memory allocation, as well as the acceleration of job execution, is concerned by modifying the fitness of resource allocation.

2 Modeling and analysis

2.1 Resource allocation model

Definition 1 Resource allocation type. Denotes $Worker = \{w_1, w_2, \dots, w_m\}$ as the set of workers, $Resource = \{r_1, r_2, \dots, r_n\}$ as a collection of resource types including CPU, memory, disk, and $r_w = (r_{w1}, r_{w2}, \dots, r_{wl})$ represents l available resource vector of worker w_m , where r_{wi} is the i th available resource in the worker w , and the i th resource in all workers could be normalized as:

$$\begin{aligned} & \text{Normalize}(r_{wi} \rightarrow (0, 1)) \\ & * rtype(cpu, memory, disk) \end{aligned} \quad (1)$$

$j = \{j_1, j_2, \dots, j_n\}$ denotes as the set of running jobs at the same time, $V_{rj} = (v_{rj1}, v_{rj2}, \dots, v_{rjk})$ represents the resource requirement vector of job j , since the resource requirement of each job is different, and the resource requirements of all jobs are represented as:

$$\begin{aligned} RV &= (V_{r1}, V_{r2}, \dots, V_{rj}) \\ &= ((v_{r11}, v_{r12}, \dots, v_{r1k}), (v_{r21}, v_{r22}, \dots, v_{r2k}), \dots, \\ &\quad (v_{rj1}, v_{rj2}, \dots, v_{rjk})); v_{rjk} \geq 0 \end{aligned} \quad (2)$$

Then, the resource requirements type for all jobs are expressed as:

$$\begin{aligned} TypeRV &= (typeRV_1, typeRV_2, \dots, typeRV_k) \\ &= (rtypes(max(v_{r11}, v_{r12}, \dots, v_{r1k}), \\ &\quad rtypes(max(v_{r21}, v_{r22}, \dots, v_{r2k}), \dots, \\ &\quad rtypes(max(v_{rj1}, v_{rj2}, \dots, v_{rjk}))) \end{aligned} \quad (3)$$

The resource requirements are submitted to the system before the execution of the job, and the jobs will be assigned to workers with idle resources that can feed their requirements. Assume $workers = \{w_1, w_2, \dots, w_m\}$ as workers dealing with task j , $v_{aj} = (v_{aj1}, v_{aj2}, \dots, v_{ajk})$ as the resource allocation vector of task i in worker w_1 . In principle, workers should strictly allocate resources in accordance with the resource requirements table, which is represented as:

$$v_{ajk} = \frac{v_{rjk}}{workerNum}, j \in jobs \quad (4)$$

2.2 Parallelism degree model

In Spark, task parallelism degree is used to measure the number of concurrent tasks, which can be specified by the user, and it could not exceed the whole instance number that equals to the product of the number of worker and the number of CPU cores in each worker.

Definition 2 Parallelism degree. Denotes the number of workers as $workerNum$, the number of CPU cores in each worker node as $coreNum$; therefore, the tasks executing concurrently is $workerNum \times coreNum$ supported by the hardware environment. If the parallelism parameter specified by the user is p_{user} , then parallelism degree $parallelismDegree$ is the minimum value of $workerNum \times coreNum$ and p_{user} :

$$parallelismDegree = \min(p_{user}, workerNum \times coreNum) \quad (5)$$

Definition 3 Idle time. It is defined to indicate idle time due to uneven task allocation. According to Definition 5, when user parallelism is greater than the hardware parallelism, that is $p_{user} = (workerNum \times coreNum)$, the number of pipelines within the stage is greater than task parallelism. Then, the worker needs to allocate task in multiple turns, and the number of turns can be expressed as:

$$turnNum = \text{ceiling}\left(\frac{p_{user}}{workerNum \times coreNum}\right) \quad (6)$$

where the result of ceiling function is the smallest integer that is greater than or equal to the value of the parameter. By formula 6, we can obtain that when l is an integral multiple of $(workerNum \times coreNum)$, all workers should execute the task in each round of distribution. If the remainder when p_{user} divides $(workerNum \times coreNum)$ is not 0, there is at least one idle node in the final round, and the number of idle workers can be expressed as:

$$\begin{aligned} idleNum = & (workerNum \\ & \times coreNum) \bmod (p_{user}, (workerNum \\ & \times coreNum)) \end{aligned} \quad (7)$$

where $\bmod(p_{user}, (workerNum \times coreNum))$ represents reminder. Due to random allocation of tasks, the probability that p_{user} is the integer times of $(workerNum \times coreNum)$ is very small, then the allocation load of tasks in the final round is likely to be uneven. Assume the set of h pipeline tasks in the final round as $Task_{pipes_{last}} = \{Task_{pipe_{l1}}, Task_{pipe_{l2}}, \dots, Task_{pipe_{lh}}\}$, where $h < (workerNum \times coreNum)$. Then, the idle time of the bye node is:

$$T_{idle_w} = \max(T_{pipe_{l1}}, T_{pipe_{l2}}, \dots, T_{pipe_{lh}}) \quad (8)$$

2.3 Allocation fitness model

Definition 4 Resource occupancy rate. Assume T_{fixed} as a measurement interval, T_{job_i} as the actual execution time of the job i . The occupancy rate of r th resources OC_{ir} is defined as the proportion of the resources used by the workers, which is expressed as:

$$OC_{ir} = (v_{r_j} \times \frac{T_{job_i}}{T_{fixed}}), r \times R \quad (9)$$

Definition 5 Allocation fitness degree. Assume workLoad as the total workload, CAs = $\{ca_1, ca_2, \dots, ca_n\}$ represents the set of computing ability of each worker in the $workers = \{w_1, w_2, \dots, w_n\}$. Thus, the mean value of the task execution time in all workers can be defined as:

$$meanValue = \frac{workLoad}{\sum_{w \in workers} ca_{w_i}} \quad (10)$$

Without considering the waiting time, the execution time of tasks in worker w_i with the task allocation amount allocationLoad_{wi} can be expressed as:

$$T_{task_{w_i}} = \frac{allocationLoad_{w_i}}{ca_{w_i}}, w_i \in workers \quad (11)$$

Therefore, the variance of task execution time is represented as:

$$variance_{w_i} = (T_{finish_{w_i}} - meanValue)^2 \quad (12)$$

The allocation fitness degree of worker w_i can be formulated as:

$$\begin{aligned} allocationFitness_{w_i} &= \frac{1}{variance_{w_i}} \\ &= \frac{1}{(variance_{w_i} - meanValue)^2} \end{aligned} \quad (13)$$

Lemma 1 For all workers involved in the calculation, the greater the allocation fitness, the shorter the execution time of the job and the higher the computational efficiency.

Proof From the point of view of task allocation, the execution time of the job can be expressed as:

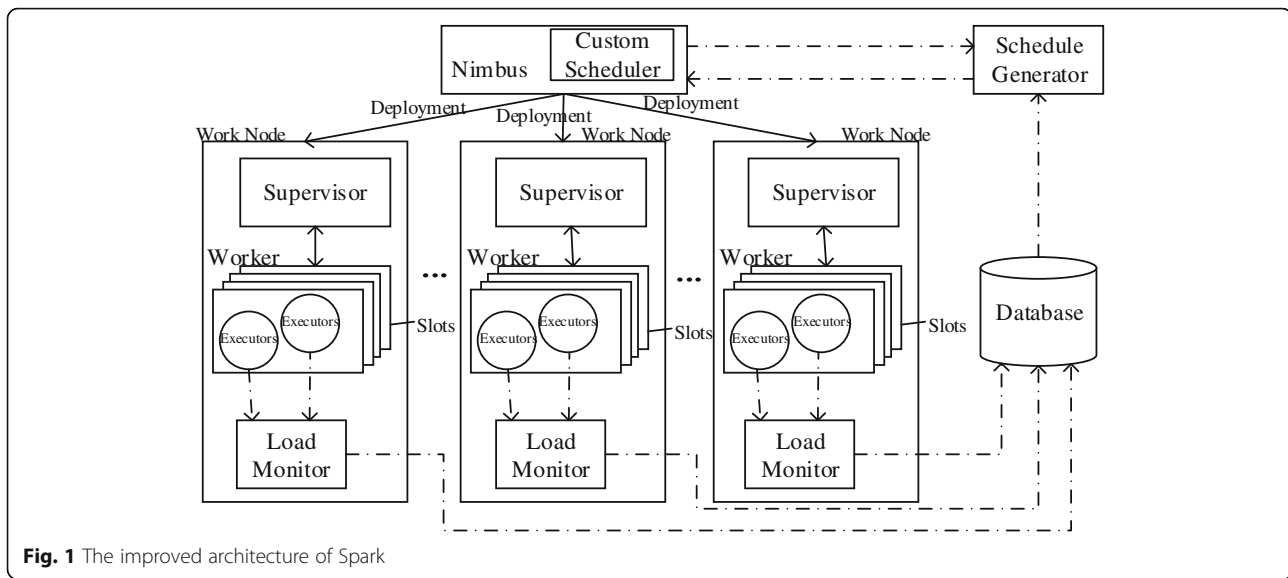
$$T_{job} = \max(T_{finish_1}, T_{finish_2}, \dots, T_{finish_n}) \quad (14)$$

According to formula, the allocation fitness is inversely proportional to the variance. If the fitness value is greater, the variance is smaller, which means the completion time of tasks in the work is closer to the mean. So, when recovery entropy takes a maximum value, job execution time is shortest and execution efficiency is the highest. Therefore, we select the worker with the higher load to immigrate the latter task to the worker with the lower load to reach a higher degree of parallelism and allocation fitness.

3 The performance optimization strategy

3.1 Construct basic data

The improved architecture of Spark with optimization strategy is shown in Fig. 1.



To deploy the performance optimization strategy in Spark, it is necessary to implement the scheduling method in the `spark.scheduler.TaskSchedulerImpl` interface. The DAG scheduler contains all the topology information of current cluster operation, including all kinds of parameter configuration information and mapping between thread and the component ID; cluster object contains all status information of the current cluster, including the mapping information between each thread, node and executor of topology, the use and information of idle workers, and slots. The above information can be obtained through the API object. The CPU occupancy information of each thread in the topology can be obtained through the `getThreadCpuTime(long id)` method in `ThreadMXBean` class of Java API, where `id` is the thread ID; network bandwidth occupancy information of each thread can be obtained by measuring each RDD size in the experiment as well as monitoring the data transmitting rate of each thread in Spark UI, then estimating by simple accumulation. Due to the threads existing shared memory, the memory occupancy of each thread can only be roughly estimated by the `-Xss` parameter in the configuration file; in addition, the hardware parameters and load information in operating system could through the `/proc` directory to access relevant documents. When the code is written, it will package jar to the `Spark_HOME/lib` directory and run after configuring `spark.scheduler` in `spark.yaml` of the master node.

3.2 Performance optimization strategy

The key problem of the optimization strategy is the selection of the destination node. However, in order to meet the requirements of the worker, it is necessary to exclude the nodes that do not meet the resource constraint model.

Denote m_s and m_d as the total amount of memory resource in the source node and the alternative destination node respectively. In the process of decision-making to assign the latter task, it is necessary to continue to move out of other tasks, until the source node resources occupied are less than the threshold. Finally, select the optimal destination node to ensure the allocation fitness reaching the larger value. It should be noted that, when the memory, disk, or network bandwidth resources overflow, the optimization strategy is the same as this section, only to calculate the corresponding type of resource.

Then, the detail steps for the process of optimization strategy are shown in algorithm 1.

- Step 1. Initialize the read data path and the number of data partitions. Spark uses RDD's text file operator to read the data from HDFS to the memory of the Spark cluster.
- Step 2. Obtain the default parallelism degree and collect statistical information to calculate data resource occupancy degree in the system.
- Step 3. The degree of parallelism and allocation fitness are updated based on the former function shown in sections 2.2 and 2.3 in combination with the data information acquired in step 2, and then, select the id of workers with the higher load.
- Step 4. Save the corresponding parameters to the database and update the information when the status of the resource changes. After selecting the source node and destination node, exchange their tasks and refresh the remaining CPU, memory, and network bandwidth resource of the source node and the destination node.
- Step 5. The TaskScheduler then selects the set of workers with the lower load to assign a task to get a larger degree of parallelism and the allocation fitness.

Algorithm 1.The performance optimization algorithm

Input : d_i, m_i , the total disk, memory capacity of node;
 du_i, mu_i , the used disk, memory capacity of node;
 $thresh$, the threshold assign by programmer

```

1   $sc \leftarrow \text{new SparkConf}(conf)$ 
2   $parallelismDegree \leftarrow spark.default.parallelism$ 
3   $\text{If } parallelismDegree \geq workernum * cores;$ 
4     $concurrentTaskNum = workerNum * cores;$ 
5   $\text{else if}$ 
6     $concurrentTaskNum = workerNum * cores;$ 
7   $\text{end if}$ 
8   $dataRDD \leftarrow sc.textFile(Filepath, partition)$ 
9   $\text{if } m_s \leq thresh \text{ then}$ 
10    $\text{/* the memory occupancy reach the threshold */}$ 
11    $\text{for each task in taskQueue do}$ 
12      $\text{for each node in resourceList do}$ 
13        $\text{if } requirement(task_i) \neq null \parallel m_i \leq node_i \parallel d_i \leq node_i$ 
14        $\text{then}$ 
15          $candidateList(task_i) \leftarrow add(node_i);$ 
16          $\text{/*add the alternative destination node with enough resources*/}$ 
17        $\text{end if}$ 
18      $\text{end for}$ 
19    $\text{end for}$ 
20    $\text{while}$ 
21      $\text{for } i=1 \text{ to } nodelist \text{ in } candidateList \text{ do}$ 
22        $node(i).allocationFitness \leftarrow (m_i/mu_i, d_i/du_i);$ 
23        $refresh(mu_i, du_i);$ 
24      $\text{end for}$ 
25      $destinationNode \leftarrow order(node(i).allocationFitness);$ 
26      $scheduleTask(destinationNode, concurrentTaskNum)$ 
27    $\text{end for}$ 
28  $\text{end while}$ 

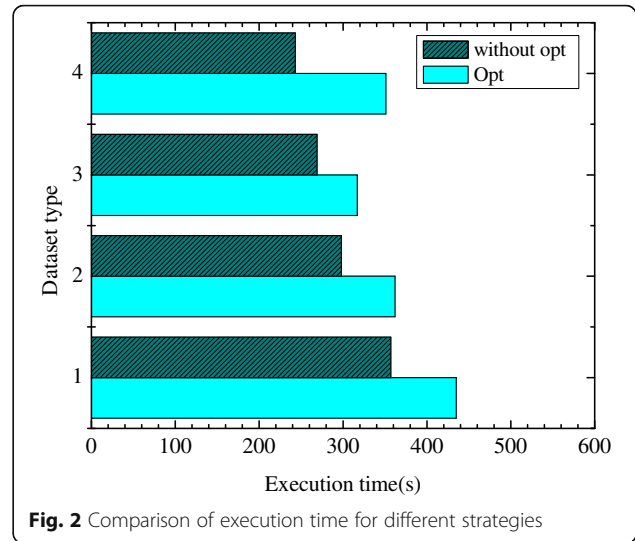
```

4 Result and discussion**4.1 Experimental platform**

We established a computing cluster by using 1 server and 8 work nodes; the server is set as Master Hadoop and NameNode Spark, and the others are set as Hadoop Slavers and Spark DataNodes. The details of the configuration are shown in Table 1. The task

Table 1 Configuration parameters

Parameters	Values
CPU	Intel CORE i7/2.2GHZ
RAM	4GB
NIC	1000Mbit/s
Hard Disk	200GB/SATA3.0(6Gbps)
OS	ubuntu 12.04
Spark	Apache Spark 2.1.0
Hadoop	Apache Hadoop 2.6
Scala	Scala-2.10.4
JDK	OpenJDK 1.8.0 25

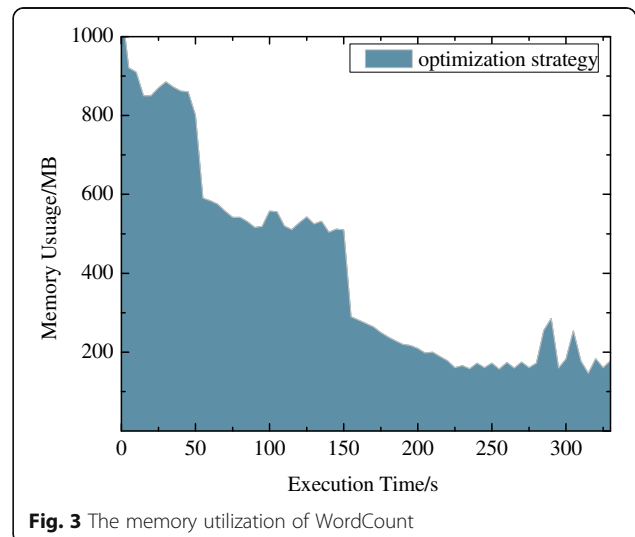
**Fig. 2** Comparison of execution time for different strategies

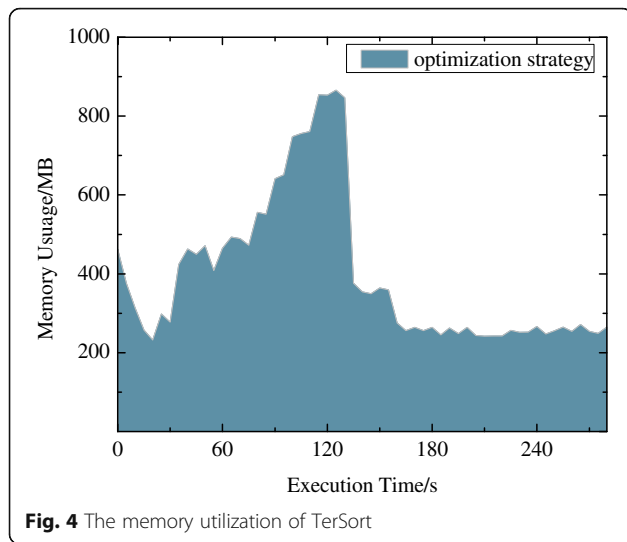
execution time is acquired from the Spark console, and nomon monitors the memory usage.

4.2 Execution time evaluation

In order to verify the algorithm in several different types of operations under the concurrent environment performance, we use the Spark official work examples to form a working set, including the type of four algorithms; dataset type 1, 2, 3, 4 denotes WordCount, TeraSort, K-Means, and PageRank as jobs. Figure 2 is a comparison of the execution time for different strategies.

Figure 2 shows that in the case of performance optimization, the recovery acceleration of the K-Means and PageRank of the proposed strategy is better than that of without the optimization strategy, which is a comparison of the operations of wide dependency in K-Means and PageRank, WordCount and TeraSort. The corresponding acceleration rate are 17.9%, 17.6%, 15.1%, and 30%

**Fig. 3** The memory utilization of WordCount



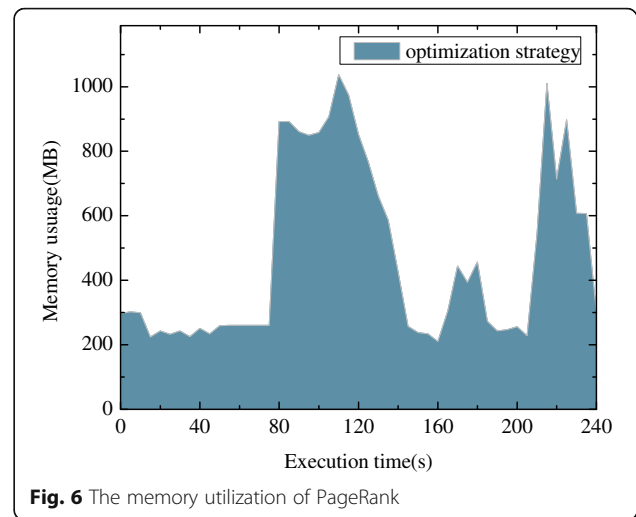
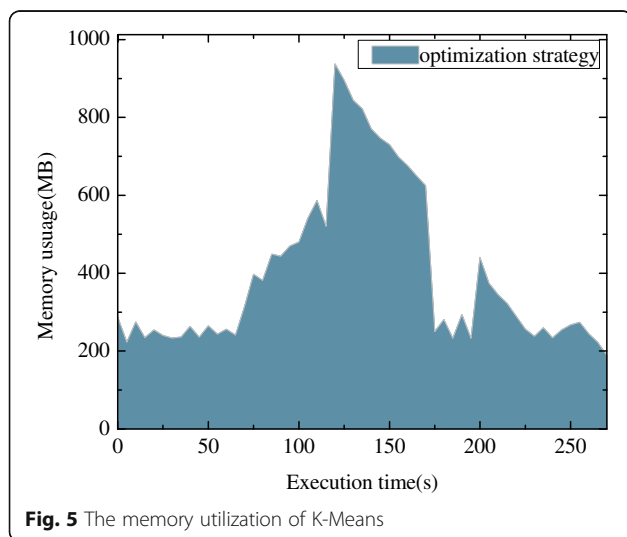
respectively. The improper parallelism degree and task allocation may induce a large amount of out of memory and increased disk I/O, which will decrease the execution efficiency and lead to higher overhead in execution time.

Thus, compared to the existing scheduling mechanism, the scheduling with performance optimization strategy can more effectively reduce the latency, and the implementation process will not have a greater impact on the performance of the cluster.

4.3 Memory utilization evaluation

Figures 3, 4, 5, and 6 are monitored under the optimization strategy proposed in this paper. Memory utilization of four different job changes during the execution of worker 3.

Memory utilization is related to the type of job and the distribution of input data. For the same algorithm, the greater the amount of data processed is, the greater the

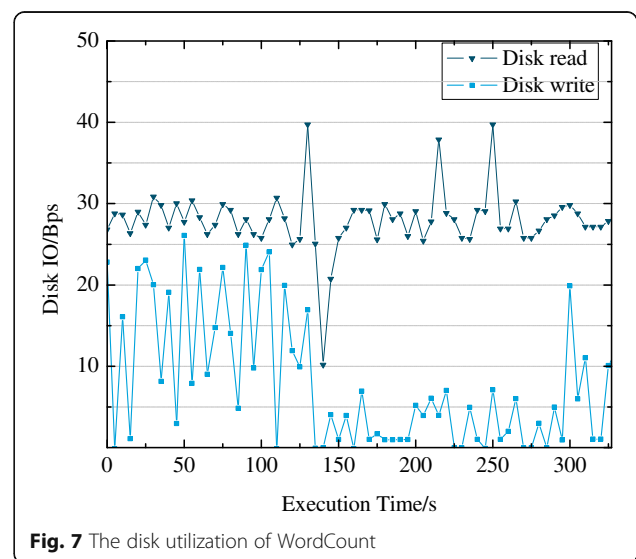


amount of memory occupied. As shown in Figs. 3, 4, 5, and 6, WordCount and TeraSort have a relatively stable memory footprint with the increase of execution time, while K-means and PageRank have different memory occupancy rates as the processing task phases are different.

4.4 Disk I/O evaluation

Similarly, the disk I/O has different characteristics as the type of job varies. Figures 7, 8, 9, and 10 are monitored under the optimization strategy proposed in this paper. The memory utilization of four different job changes during the execution of worker 3.

As far as the disk I/O rate is concerned for the task processing the data from the local disk, the corresponding local data reads on a worker will be generated, and a certain disk I/O is consumed. If the network data is processed, additional network I/O is also produced



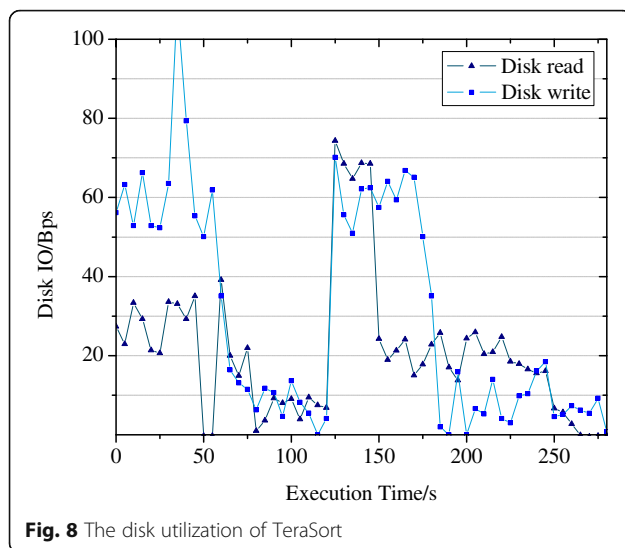


Fig. 8 The disk utilization of TeraSort

because the worker needs to read data from the remote disk, and memory outage may produce more frequent disk I/O. As it is known in Figs. 7, 8, 9, and 10, disk I/O of WordCount is more obvious, and the other three jobs are lower. At the beginning of execution for K-Means and TeraSort, disk I/O is significantly increased because the task is assigned to worker 3, and it needs to read some data from the disk at this time.

5 Conclusions

In this paper, our contributions can be summarized as follows. First, we analyze a theoretical relationship of degree of parallelism and allocation fitness. Second, we propose an evaluation model that is pluggable for task assignment. Third, on the basis of the evaluation model, the strategy can take resource characteristics into consideration and assign tasks to the worker with a lower

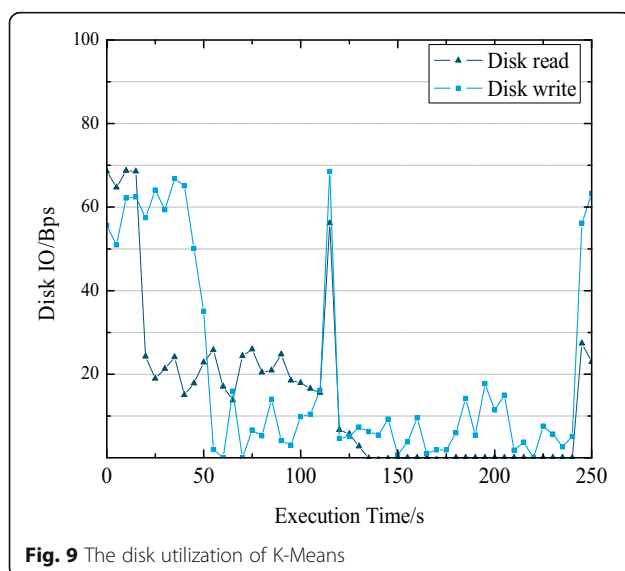


Fig. 9 The disk utilization of K-Means

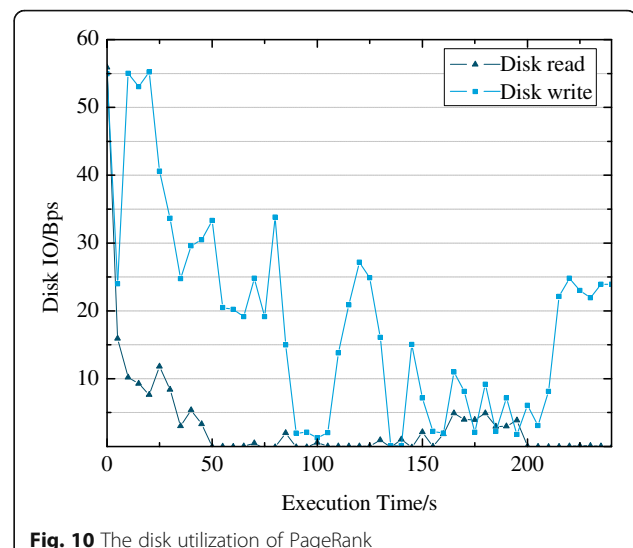


Fig. 10 The disk utilization of PageRank

load to increase execution efficiency. Numerical analysis and experimental results verified the effectiveness of the presented strategy.

Our future work is mainly concentrated on analyzing the general principles of the requirements for different types of operating resources for in-memory computing framework and design the optimization strategy adapting to the load and type of jobs.

Abbreviations

CPU: Central processing unit; DAG: Directed acyclic graph; IMC: In-memory computing; SAP: System applications and products; SIMD: Single instruction, multiple data; SQL: Structured query language

Acknowledgements

The authors would like to thank the reviewers for their thorough reviews and helpful suggestions.

Funding

This paper was supported by the National Natural Science Foundation of China under Grant No. 61262088, 61462079, 61562086.

Availability of data and materials

All data are fully available without restriction.

Authors' contributions

CTY is the main writer of this paper. She proposed the main idea, completed the experiment, and analyzed the result. CYY and CB gave some important suggestions for this paper. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details

¹School of Mechanical and Electrical Engineering, Shaoxing University, Shaoxing 312000, People's Republic of China. ²School of Information Science and Engineering, Xinjiang University, Urumqi 830008, People's Republic of China.

Received: 24 July 2018 Accepted: 18 September 2018

Published online: 16 October 2018

References

1. S. John Walker, Big data: a revolution that will transform how we live, work, and think. *Int. J. Advert.* **33**(1), 181–183 (2014)
2. K. Kambatla, G. Kollias, V. Kumar, et al., Trends in big data analytics[J]. *J. Parallel Distrib. Comput.* **74**(7), 2561–2573 (2014)
3. C.L.P. Chen, C.Y. Zhang, Data-intensive applications, challenges, techniques and technologies: a survey on big data. *Inf. Sci.* **275**(11), 314–347 (2014)
4. F. Provost, T. Fawcett, Data science and its relationship to big data and data-driven decision making. *Big Data* **1**(1), 51–59 (2013)
5. C. Napoli, G. Pappalardo, E. Tramontana, A mathematical model for file fragment diffusion and a neural predictor to manage priority queues over BitTorrent. *Int. J. Appl. Math. Comput. Sci.* **26**(1), 147–160 (2016)
6. Y. Lamari, S.C. Slaoui, Clustering categorical data based on the relational analysis approach and MapReduce. *J. Big Data* **4**(1), 28 (2017)
7. C. Cho, S. Lee, Effective five directional partial derivatives-based image smoothing and a parallel structure design[J]. *IEEE Trans. Image Process.* **25**(4), 1617–1625 (2016)
8. Kim W S, Lee J, An H M, et al. Image Filter Optimization Method based on common sub-expression elimination for Low Power Image Feature Extraction Hardware Design. *10*(2), 192–197 (2017)
9. H. Seol, W. Shin, J. Jang, et al., Elaborate refresh: a fine granularity retention management for deep submicron DRAMs. *IEEE Trans. Comput.* **99**, 1–1 (2018)
10. Y. Tang, B. Gedik, Autopipelining for data stream processing. *IEEE Trans. Parallel Distrib. Syst.* **24**(12), 2344–2354 (2013)
11. I. Jo, D.H. Bae, A.S. Yoon, et al., YourSQL: a high-performance database system leveraging in-storage computing. *Proc. of the Vldb Endowment* **9**(12), 924–935 (2016)
12. R. Wu, L. Huang, P. Yu, et al., SunwayMR: a distributed parallel computing framework with convenient data-intensive applications programming. *Futur. Gener. Comput. Syst.* **71**, 43–56 (2017)
13. B. Sengupta, A. Das, Use of SIMD-based data parallelism to speed up sieving in integer-factoring algorithms[J]. *Appl. Math. Comput.* **293**(1), 204–217 (2017)
14. I.P. Egwuotuoha, D. Levy, B. Selic, S. Chen, A survey of fault tolerance mechanisms and checkpoint/restart implementations for high-performance computing systems [J]. *J. Supercomput.* **65**(3), 1302–1326 (2013)

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► [springeropen.com](https://www.springeropen.com)