Data Reduction Strategies for Load Balancing in Data-Intensive Workflows in Cloud Environments (Postprint)

Authors: Hu Zhigang, Li Jia, Meiguang Zheng

Date: 2018-05-24T00:00:00+00:00

Abstract

The efficient and reasonable scheduling of data-intensive workflow applications has emerged as a critical challenge urgently requiring solution in cloud computing. To address this issue, we first construct a directed hypergraph model for data-intensive workflows. We then propose the concept of data support capability and reduce the model through merge operations based on this capability. Finally, by optimizing the hypergraph multilevel partitioning algorithm, we propose HEFT-P, a data-intensive workflow scheduling strategy with data reduction. Experimental results demonstrate that HEFT-P, compared with typical workflow scheduling strategies such as HEFT, CPOP, and MCP, can effectively reduce and optimize data-intensive workflows, achieving shorter scheduling times.

Full Text

Preamble

Data Reduced Strategy for Load-Balanced Data-Intensive Workflow in Clouds

Hu Zhigang, Li Jia, Zheng Meiguang
(College of Software Engineering, Central South University, Changsha 410075, China)

Abstract: How to schedule data-intensive workflow applications efficiently and reasonably has become one of the key issues in cloud computing. To address this issue, we first construct a directed hypergraph model for data-intensive workflow. We then propose the concept of data supportive ability and perform model reduction through merge operations based on this capability. Finally, by optimizing the hypergraph multi-level partitioning algorithm, we propose

a data-reduced scheduling strategy called HEFT-P for data-intensive workflow. Simulation results demonstrate that compared with typical workflow scheduling strategies HEFT, CPOP, and MCP, HEFT-P can effectively reduce and optimize data-intensive workflow, achieving shorter scheduling lengths.

Keywords: data-intensive workflow; directed hypergraph; data reduced scheduling; cloud computing; load balancing

0 Introduction

Data-intensive computing has emerged as a promising new paradigm that aims to better understand and refine problem-solving domains through the analysis of massive datasets. Cloud computing' s on-demand access, scalability, and availability for running large-scale applications in virtualized environments provide crucial support for big data storage, management, and analysis. Large-scale network applications based on cloud computing exhibit distributed, heterogeneous characteristics and a data-intensive trend, exemplified by scientific workflow systems. These applications, known as Data-Intensive Applications, typically process data at the TB or even PB scale.

Tasks in data-intensive applications must acquire, process, and transmit enormous amounts of data. Inefficient data selection and task scheduling strategies lead to excessive data transmission and access volumes, which not only increase user costs for cloud resources but also severely impact the execution efficiency of scientific workflows. Therefore, how to reduce and optimize data-intensive workflows, simplify big data representation, achieve on-demand reduction, and obtain better knowledge abstraction with lower complexity has become a valuable research topic. This work aims to perform workflow data reduction and task scheduling for data-intensive applications in cloud environments, reducing workflow scheduling time by minimizing redundant data transmission and inter-task communication.

An accurate task scheduling model forms the foundation for problem investigation, aiding researchers in understanding and analyzing problems to better solve them. Current task scheduling models primarily use task priority graph (TPG) and directed acyclic graph (DAG) to represent dependency relationships. In DAG, each vertex represents a task in the application, and each edge represents data dependency between tasks. However, with the emergence of data-intensive applications in cloud environments, relationships between tasks have become more complex. To more accurately represent actual task relationships, this paper adopts directed hypergraph to model data-intensive workflow. A hyperedge in a directed hypergraph can connect multiple nodes, effectively representing asymmetric dependency relationships and clearly expressing many-to-many relationships between tasks. Beyond concepts from general graph theory, hypergraphs possess capabilities such as hyperedge merging and splitting that ordinary graphs lack. Hypergraph partitioning, as a typical combinatorial opti-

mization problem in graph theory, has wide applications in VLSI circuit design, parallel computing, image recognition, and other fields. Due to the superiority of hypergraph partitioning, the academic community has proposed numerous excellent partitioning methods, including multi-level algorithms, spectral methods, KL/FM (Kernighan Lin/Fiduccia Mattheyses), and various local optimization algorithms. Corresponding graph partitioning software packages have also emerged, such as hMETIS, PaToH, and the parallel partitioning software Zoltan toolkit proposed by Devine et al., which enables more efficient hypergraph partitioning.

Some scholars have leveraged these hypergraph advantages to solve practical problems. Sun Xuedong et al. proposed a directed hypergraph-based workflow resource allocation balancing optimization method that optimizes enterprise processes using hypergraph properties and formal rules for interaction between activity capability requirement sets and process structures. Laura et al. used directed hypergraph to model the relationship between learning activities and their associated component capabilities, adjusting relevant algorithms and constructing a large-scale resource library of learning components.

Currently, data-intensive applications have been widely used in astronomy, high-energy physics, and other fields, making their efficient and reasonable scheduling a research hotspot in cloud computing. Classical dependency task scheduling algorithms such as Heterogeneous Earliest Finish Time (HEFT) and Critical Path on a Processor (CPOP) select tasks based on priority and assign them to appropriate processors, achieving relatively short scheduling times. However, these algorithms fail to produce good results when the number of processors is excessive or in simple cases. Xiao Peng et al. introduced virtual data storage nodes into traditional DAG graphs and proposed an energy-aware scheduling strategy for data-intensive workflow that reduces task execution energy consumption from an energy-aware perspective. Yuan Yingchun et al. analyzed the parallel and synchronous completion characteristics of activities in DAG graphs, transformed workflow deadlines into layer deadlines, and proposed a deadline-constrained reverse layered cost optimization algorithm that optimizes costs within time constraints. Ahmad et al. proposed a Hybrid Genetic Algorithm (HGA) that introduces heuristic strategies in the initial population to provide direction for finding optimal solutions, enabling rapid convergence in short time, though the algorithm has high time complexity. Lin et al. proposed a cost-driven workflow scheduling strategy with deadline constraints in multi-cloud environments that introduces local critical path theory for holistic allocation of workflow subtasks to compress data communication and reduce execution costs, but this strategy ignores factors such as inter-cloud data communication costs and instance start/stop times. Mon et al. proposed a task clustering method based on task dependency relationships in cloud environments (CMTD) to reduce execution overhead and improve computational granularity of scientific workflow tasks, but this algorithm does not consider heterogeneous computing resources, which does not match real cloud computing environments.

In summary, this paper proposes a hypergraph-based data reduction scheduling strategy in cloud environments to reduce communication volume between computing nodes and workflow completion time. The overall approach is as follows: (a) Abstract data files according to characteristics of data-intensive applications, propose the concept of data supportive ability, and construct a directed hypergraph model considering data supportive ability to better express relationships between tasks; (b) Merge tasks based on data supportive ability, optimize hypergraph coarsening strategy, reduce and partition the model to minimize communication volume between computing nodes; (c) Based on this, propose the task scheduling algorithm HEFT-P, where the total instruction length of each task subset is proportional to the instruction execution speed of its computing node, enabling the workflow to obtain shorter completion time under load balancing constraints.

1 Directed Hypergraph Model for Data-Intensive Workflow

1.1 Model Definition

In a data-intensive application, the problem-solving approaches of two tasks may differ, but the data elements required for their processing may be stored in the same data file. This inevitably leads to the same file being requested by multiple tasks and transmitted to the computing nodes where tasks reside. This section utilizes the correspondence between hypergraph properties and data-intensive workflow properties to construct a directed hypergraph model for data-intensive workflow based on relationships between files and tasks.

Let $H = (V, E)$ be a directed hypergraph model for data-intensive workflow, where V is the set of all nodes divided into two categories: D representing dataset nodes and T representing task nodes, with $V = D \cup T$ and $D \cap T = \emptyset$. E is the set of hyperedges, also divided into two categories: X representing data supportive edges from datasets to tasks, and Y representing workflow hyperedges for task dependencies, with $E = X \cup Y$ and $X \cap Y = \emptyset$. w represents node weights, and c represents hyperedge weights.

For the task set $T = \{t_1, t_2, ..., t_n\}$, where t_i is a task node in the data-intensive application, let $pred(t_i)$ denote the predecessor set of t_i and $succ(t_i)$ denote the successor set of t_i. A task t_i without predecessors is called an entry node, denoted as t_{entry}, and a task without successors is called an exit node, denoted as t_{exit}. Based on this, we define workflow hyperedges.

Definition 1 (Workflow Hyperedge). $y \in Y$ is defined as $y = (T(y), H(y))$, consisting of the current task $T(y)$ pointing to its successor tasks $H(y)$, i.e., $T(y) \subseteq T$ and $H(y) \subseteq T$. $T(y)$ is the tail node set of hyperedge y, and $H(y)$ is the head node set. For $\forall t_i \in T(y)$ and $\exists t_j \in H(y)$, t_i and t_j are operationally adjacent tasks, denoted as $t_i \rightarrow t_j$.

To efficiently select appropriate data for tasks and better analyze the functions that data can achieve for knowledge abstraction, this paper adds identification

of data supportive ability, defined as follows:

Definition 2 (Data Supportive Ability). In a data-intensive application, the historical functions achieved by input data obtained through the NWS service are recorded as data supportive ability a_k, and this marker is assigned to its corresponding input file, where k is the identifier of data supportive ability, i.e., the function type that the data can achieve. All p types of data supportive abilities in the application constitute the capability type set $A = \{a_1, a_2, \dots, a_p\}$, where p is the total number of capability types in the data supportive ability type set.

For the input file set $D = \{d_1, d_2, \dots, d_m\}$ in a data-intensive application, where d_j is a dataset file (referred to as dataset node below), we define data supportive hyperedges based on this.

Definition 3 (Data Supportive Hyperedge). $x \in X$ is defined as $x = (D(x), T(x))$, pointing from dataset node $d_j \in D(x)$ to tasks $t_i \in T(x)$ that use it as input data, i.e., $D(x) \subseteq D$ and $T(x) \subseteq T$. d_j can be mapped with data supportive abilities in A to form the data supportive ability set S_j that d_j possesses. All S_j of d_j and A together constitute the dataset supportive ability matrix (DM) of size $m \times p$, where m is the number of dataset nodes. There also exists a mapping relationship between t_i and A, forming the three-dimensional dataset-task ability matrix (DTAM) as shown in [Figure 1: see original paper]. For example, $\Gamma(d_j, a_k, t_i) = 1$ indicates that d_j can provide supportive ability a_k required for t_i execution.

All R_i of tasks t_i together constitute the task requirement ability matrix (TM) of size $n \times p$, where n is the number of tasks. This paper constructs a DTAM-based directed hypergraph model by integrating DM and TM.

Based on the above definitions, we describe the nodes in the directed hypergraph model. Let t_i be a task node with encoding i, I_i be the input data for t_i execution, and O_i be the output data after t_i completion. The supportive dataset for t_i is denoted as $S_i^d = \{d_j \mid \Gamma(d_j, a_k, t_i) = 1, d_j \in D, a_k \in A\}$. Let d_j be a dataset node with encoding j, and its supportive ability set be $S_j^a = \{a_k \mid \Gamma(d_j, a_k, t_i) = 1, t_i \in T, a_k \in A\}$.

Taking [Figure 2: see original paper] as an example, dataset nodes d_1 and d_2 have supportive abilities $S_1^a = \{a_1, a_2\}$ and $S_2^a = \{a_1, a_2\}$, respectively. Tasks t_3 and t_5 require data abilities $R_3 = \{a_1\}$ and $R_5 = \{a_2\}$, respectively. From the DTAM matrix, we obtain $\Gamma(d_1, a_1, t_3) = 1$, $\Gamma(d_1, a_2, t_5) = 1$, $\Gamma(d_2, a_1, t_3) = 1$, and $\Gamma(d_2, a_2, t_5) = 1$. Therefore, the supportive datasets for t_3 and t_5 are $S_3^d = \{d_1, d_2\}$ and $S_5^d = \{d_1, d_2\}$, respectively.

This paper considers the relationships among tasks, data, and data supportive abilities in data-intensive workflow. Based on the DTAM matrix, we obtain [Figure 2: see original paper], a directed hypergraph model for data-intensive workflow considering data supportive ability, where circles represent tasks and hexagons represent input files. Through this mapping, we transform the schedul-

ing problem of data-intensive workflow into a data reduction and partitioning problem for directed hypergraphs.

When operating on the hypergraph model of data-intensive workflow, we adopt hypergraph partitioning theory as the primary guidance. Given a directed hypergraph $H = (V, E)$ and an integer k, we partition H into k node subsets with approximately equal weights, i.e., $Partition = \{p_1, p_2, ..., p_k\}$, where $\bigcup_{i=1}^{k} p_i = V$ and $p_i \cap p_j = \emptyset$ for $i \neq j$. The weight of partition p_i is $w_{p_i} = \sum_{t \in p_i} w_t$. The partitioning must satisfy the following conditions: (a) $\frac{w_{p_i}}{W_{avg}} \leq 1 + \varepsilon$, where $W_{avg} = \frac{\sum_{t \in V} w_t}{k}$ is the average weight and ε is a predefined load balancing threshold; (b) If nodes belonging to hyperedge e belong to at least two different partitions p_i and p_j, then e is said to be cut. We attempt to minimize the connectivity $\lambda - 1$, i.e., the number of cut edges, where λ is the number of partitions connected by hyperedge e. Minimizing connectivity corresponds to minimizing communication volume between different virtual machines under load balancing constraints.

1.2 Problem Description

Based on the above model definition for data-intensive workflow, the scheduling time (makespan) is determined by the critical path (CP) in the hypergraph model, i.e., the longest path in the hypergraph. We adopt the calculation method from reference [4] as follows:

The upward level $tlevel(t_i)$ represents the longest path from the entry node to t_i, i.e., the time required from the entry task to the current task' s start, excluding t_i' s weight. The downward level $blevel(t_i)$ represents the longest path from t_i to the exit node. Their calculation formulas are:

$$tlevel(t_i) = \max_{t_j \in pred(t_i)} \{tlevel(t_j) + w_j + c_{ji}\}$$

$$blevel(t_i) = \max_{t_j \in succ(t_i)} \{blevel(t_j) + w_j + c_{ij}\}$$

where w_i is the weight of task t_i (i.e., computational workload) and c_{ij} is the communication volume between task t_i and its successor tasks. As seen from these formulas, the computational workload of critical tasks on the critical path cannot be changed. This paper reduces task scheduling time by merging critical tasks or assigning them to the same computing node to enhance locality, thereby reducing inter-task communication volume.

2 Hypergraph Partitioning Scheduling Algorithm for Data Reduction

This section optimizes the structure of the directed hypergraph model through support capability-based data reduction and hypergraph partitioning algorithms to reduce total task communication volume and consequently decrease task scheduling time. Support capability-based data reduction is achieved through task merging operations, thereby reducing repeated transmission of identical data. Hypergraph partitioning is an NP-hard problem that can be effectively solved using heuristic methods. This paper employs a multi-level algorithm to partition the data-intensive workflow hypergraph model and optimizes the partitioning algorithm for better performance.

2.1 Support Capability-Based Merging

In data-intensive workflow, tasks often have very large input and output data volumes. However, some adjacent tasks share common supportive data. If assigned to the same virtual machine, the time consumption caused by waiting for data transmission can be reduced. Based on the relationship between tasks and supportive abilities, this paper obtains task supportive datasets from the DTAM matrix and performs data reduction on these tasks according to the following merging rules to reduce repeated transmission of identical supportive data, thereby decreasing task execution time and algorithm complexity.

Rule 1 (Merging). Tasks t_i and t_j can be merged if they satisfy: (1) t_i and t_j are operationally adjacent tasks; (2) The intersection of their supportive datasets is non-empty, i.e., $S_i^d \cap S_j^d \neq \emptyset$; (3) The sum of the two tasks' weights is less than the predetermined maximum load. If the merged task is t_c, then $I_{t_c} = I_{t_i} \cup I_{t_j} - O_{t_i} \cap O_{t_j}$ and $O_{t_c} = O_{t_i} \cup O_{t_j} - O_{t_i} \cap O_{t_j}$. The supportive dataset after merging is $S_{t_c}^d = S_{t_i}^d \cap S_{t_j}^d$.

Based on the above merging rules, we employ the Task Merging Algorithm (SMA) to perform merging operations on tasks. The SMA is described as follows:

Algorithm 1 Task Merging Algorithm (SMA)
Input: DTAM, $H = (V, E)$
Output: Merged hypergraph $H' = (V', E')$

1. begin
2. for each $t_i \in T$ do // Find supportive dataset for each task
3. `$S_i^d \leftarrow \emptyset$, $R_i \leftarrow \{a_k\}$`
4. `for each $d_j \in D$ do`
5. `  if $\Gamma(d_j, a_k, t_i) = 1$ then`
6. `    $S_i^d \leftarrow S_i^d \cup \{d_j\}$`
7. `  end if`

8. end for
9. end for
10. for each $t_i, t_j \in T$ do // Determine if tasks need merging
11. if $(t_i, t_j) \in E \land S_i^d \cap S_j^d \neq \emptyset$ then
12. if $(1+\varepsilon) \times (w_i + w_j) < W_{avg}$ then
13. Combine(t_i, t_j) // Merge according to Rule 1
14. end if
15. end if
16. end for
17. end

Lines 2-9 find the supportive dataset for each task in the workflow, initializing S_i^d as empty and assigning each task' s required R_i to it, then merging dataset nodes d_j that satisfy $\Gamma(d_j, a_k, t_i) = 1$ into set S_i^d. Lines 10-16 determine whether tasks need merging: if t_i and t_j are operationally adjacent tasks, their supportive datasets have non-empty intersection, and the sum of the two tasks' weights is less than the predetermined maximum load, then t_i and t_j are merged according to Rule 1. The weight of the merged node is the sum of the merged nodes'weights. The algorithm' s time complexity is $O(n \times m)$, where n is the number of tasks and m is the number of data files.

2.2 Multi-Level Partitioning Algorithm for Support Data

After merging tasks using the SMA algorithm, this paper employs a multi-level partitioning algorithm to partition the hypergraph model, achieving the goal of minimizing communication volume between computing nodes. The multi-level partitioning algorithm generally consists of three phases: coarsening, partitioning, and refinement. In the coarsening phase, node matching and merging operations are performed iteratively until termination conditions are met, constructing multi-level coarse hypergraphs that effectively reduce the number of hyperedges and nodes in the original hypergraph. In the partitioning phase, the smallest coarse hypergraph obtained from coarsening is initially partitioned. In the refinement phase, the initial partition of the smallest coarse hypergraph is projected back to the original hypergraph, and the hypergraph partition is optimized at each projection level, effectively solving the problem of migrating node groups in the original hypergraph. This section optimizes the coarsening phase of the multi-level partitioning algorithm.

2.2.1 Hypergraph Coarsening Based on Support Data After performing node merging operations, we proceed with hypergraph coarsening. In the coarsening phase, the original directed hypergraph $H = (V, E)$ will be represented by increasingly smaller directed hypergraphs $\{H_i = (V_i, E_i) \mid i =$

$1, 2, ..., init\}$, where $|V_{init}| < ... < |V_i| < |V_{i-1}| < ... < |V_1| < |V_0|$. Nodes in V_i are matched and merged using different matching strategies to form the next smaller coarse hypergraph V_{i+1}. The weight of the merged node is the sum of the merged nodes' weights.

However, the selection of hypergraph coarsening strategy is crucial to the quality of the coarse partitioning. Existing greedy matching algorithms primarily consider hyperedge weights, attempting to match as many nodes as possible in hyperedges with larger weights, without considering the impact of node weights on matching effectiveness and coarse partitioning results. To achieve better coarsening effects, this paper optimizes node matching strategies by considering the influence of both node and hyperedge weights on coarsening effectiveness. The proposed Coarsening Strategy Based on Support Data (CBSD) process is shown in [Figure 3: see original paper].

First, we use a sorting function $order(e)$ to sort task hyperedges in descending order. Nodes in hyperedges are then matched sequentially according to this order, enabling priority matching of nodes with smaller weights but belonging to hyperedges with larger edge weights. This approach hides edge weights as much as possible while preventing excessive node weights after merging, which benefits computing the initial coarse partition and improving refinement algorithm efficiency. Based on this analysis, the sorting function is defined as:

$$order(e) = \frac{\sum_{v \in e} w_v}{|e|}$$

The coarsening operation terminates when the node reduction rate r satisfies the termination condition θ, typically when $r < 10\%$.

2.2.2 Coarsening Partitioning and Refinement Stage When the hypergraph becomes sufficiently small (i.e., the reduction rate is below the threshold), we enter the coarsening partitioning stage. The purpose of this stage is to obtain an initial partition result for the original hypergraph by partitioning the smallest available hypergraph from the previous stage. However, partitioning the coarsened graph into k parts does not necessarily represent a high-quality partition of the original graph, but repeated local optimization operations in the refinement stage can effectively address this issue. The main goal of the refinement stage is to reduce connectivity between partitions while satisfying load balancing constraints among virtual machines. We select the optimal partition result from the coarsening stage and project it back to the original hypergraph through multiple iterations, using local optimization algorithms to improve its quality at each iteration. Successful refinement methods are primarily based on optimizations and extensions of the Kernighan-Lin (KL) and Fiduccia-Mattheyses (FM) heuristic algorithms.

2.3 Scheduling Algorithm

This paper develops the HEFT-PS scheduling algorithm based on the HEFT algorithm. The algorithm schedules tasks in units of partitions p_i obtained from the previous section, achieving efficient scheduling of data-intensive workflow. Task priority $rank_u(t_i)$ is calculated as shown in Equation (6). Traditional algorithms schedule individual tasks directly based on task priority, whereas the HEFT-PS scheduling algorithm proposed in this paper schedules tasks in task set units. Let p_{mv} be the partition with the highest priority in P, and to satisfy task execution dependency relationships, the algorithm sets the priority of p_{mv} to enable scheduling of ready tasks as soon as possible, thereby reducing task waiting time. Specifically, the largest $rank_u(t_i)$ value among tasks in p_{mv} is used as the priority of p_{mv}. We schedule p_{mv} according to this priority. After HEFT-PS scheduling, each p_{mv} is assigned to an appropriate virtual machine.

Algorithm 3 HEFT-PS Scheduling Algorithm
Input: Partition set $P = \{p_1, p_2, \dots, p_k\}$
Output: Task execution results

1. begin
2. Calculate $rank_u(t_i)$ for all tasks
3. Sort all p_i in P in non-increasing order of $rank_u(p_i)$ into scheduling list
4. while there are unscheduled partitions in the list do
5. `Take the first $p_{mv}$ from the scheduling list`
6. `for each computing node $q$ do`
7. `  Calculate $EFT(p_{mv}, q)$ on $q$`
8. `end for`
9. `Schedule $p_{mv}$ to the computing node with minimum $EFT$`
10. end while
11. end

Lines 2-3 calculate the priority of each partition (task set) and place them in the scheduling list in non-increasing order. Lines 4-10 sequentially select unscheduled partitions p_{mv}, calculate their Earliest Finish Time (EFT) on each computing node, and schedule them to the computing node with the minimum EFT. After scheduling, each task executes sequentially on its assigned computing node. The algorithm' s time complexity is $O(n \times p)$, where p is the number of computing nodes. The time complexities of HEFT and CPOP algorithms are $O(n^2 \times p)$, while MCP' s time complexity is $O(n^2 \log n)$. Since p is generally small compared to n, the advantage of the HEFT-PS algorithm becomes increasingly evident as n grows.

3 Experiments

We conduct experiments using the CloudSim cloud computing simulation platform from the University of Melbourne' s Grid Laboratory. To verify the performance of the HEFT-P scheduling algorithm, data-intensive workflow application DAG graphs are generated through both simulated settings and real workflow applications. We develop a data-intensive application (DIA) integrated test platform that can automatically customize and generate data-intensive applications, verify whether they satisfy data-intensive characteristics, and store the generated applications upon verification.

IOzone is a classic testing tool for data-intensive tasks and file systems, primarily used to test file system read/write performance. Each iozone operation is an independent test task with atomicity. Based on three core atomic operations in iozone (R, W, RW), the platform automatically combines and generates a typical data-intensive application called IOZones. Users can input main parameters of IOZones, including maximum node count, minimum node count, DAG graph layers, maximum width, and number of RW tasks, to customize IOZones applications. The generated data-intensive applications are saved locally as DAG graphs in two files: one storing DAG graph information of dependencies between DIA nodes (iozones_dag.txt) and another storing command information for each node (iozones_command.txt).

Real workflows use standard scientific workflows CyberShake and Montage. Simulated settings use task counts of {50, 100} and {25, 100} respectively, reflecting workflow application scale. Workflow instances are selected from benchmark instances in the DIA integrated test platform' s instance library. The Communication-to-Computation Ratio (CCR) represents the ratio of average task communication time to average execution time. When CCR $>$ 1, the task set is communication-intensive, i.e., data-intensive. In experiments, we set CCR $>$ 1. Thirty different instances are generated for each scale. Simulated computing node counts are {4, 8, 16, 32, 64}, and load balance thresholds are {10%, 20%, 30%, 40%, 49%}. Combining these parameters, a total of (3×30 + 2×2)×5×5 = 2350 test instances are generated. We compare the proposed scheduling algorithm with classical HEFT, CPOP, and MCP algorithms using performance metrics of Average Schedule Length (ASL) and average resource utilization. To ensure result reliability, we take the average ASL from 500 runs as test results, with all experimental data within their 95% confidence intervals.

Definition 4 (Average Resource Utilization Ratio, ARUR). ARUR is calculated as:

$$ARUR = \frac{FTV}{makespan}$$

where FTV is the average load on computing nodes, defined as the optimal completion time of total tasks, i.e., total instruction length divided by the sum

of instruction execution speeds of computing nodes. ARUR ranges in [0, 1], with values closer to 1 indicating better load balancing performance of the scheduling algorithm.

The experimental hardware environment is Intel(R) CoreTM i5 CPU 1.60GHz with 4.00GB RAM; software environment is Ubuntu 16.04.2 LTS, gcc version 5.4.0, and qt version 5.6.2.

3.1 Average Task Scheduling Length

[Figure 4: see original paper] shows the x-axis representing the number of computing nodes (increasing from 4 to 64) and the y-axis showing ASL for different algorithms scheduling the real workflow CyberShake with task counts of 50 and 100. When the number of computing nodes is small, our algorithm' s ASL is comparable to HEFT and CPOP, but as the number of computing nodes increases, our strategy consistently produces the lowest ASL.

As shown in [FIGURE:4(a)], when the number of tasks is smaller than the number of computing nodes, the ASL differences among algorithms are relatively small. Since system resources are relatively abundant, each scheduling algorithm can make scheduling decisions based on simple heuristic strategies, making the total available resources the main factor affecting ASL rather than the algorithm itself. In experiments, we found that setting the load balance threshold parameter to 0.2 during model partitioning yields the best scheduling performance for our strategy. [FIGURE:4(b)] shows that as the number of computing nodes increases, HEFT-P still achieves good scheduling results. In CyberShake, there is a task called Seismogram-Synthesis with an actual average data read volume of 547MB/job, but its required input data size is only 150MB-250MB, meaning some input data is repeatedly read, causing I/O resource waste and scheduling time extension. Compared with other algorithms, our strategy achieves lower ASL.

[Figure 5: see original paper] shows ASL for different algorithms scheduling Montage workflow with task counts of 25 and 100. Results show that as the number of computing nodes increases, ASL for all four algorithms decreases to varying degrees, with HEFT-P consistently producing the lowest ASL. Experimental results reveal that our strategy performs better on CyberShake than on Montage workflow. Since CyberShake has higher CCR values than Montage, indicating it is more data-intensive, our strategy is more suitable for scheduling data-intensive workflows.

3.2 Load Balancing Comparison

[Figure 6: see original paper] shows the average resource utilization of different algorithms under varying task counts with fixed computing nodes. The x-axis represents task count (increasing from 200 to 1000), and the y-axis shows average resource utilization. The HEFT algorithm does not consider resource utilization

objectives in workflow scheduling. As seen in the figure, as task count increases, the resource utilization of HEFT-P algorithm surpasses that of HEFT algorithm.

The above experimental results demonstrate that in cloud computing environments, as the scale of data-intensive application tasks increases, dependency relationships between tasks become more complex, and task communication overhead and corresponding scheduling length increase significantly. Compared with HEFT, CPOP, and MCP algorithms, our strategy can effectively reduce inter-task communication volume and achieve superior scheduling performance.

4 Conclusion

This paper analyzes challenges encountered when executing data-intensive applications on cloud platforms and proposes a hypergraph partitioning-based HEFT-P scheduling algorithm for data-intensive tasks. Experimental results prove that our strategy is an effective data-intensive workflow scheduling strategy in cloud computing environments. Compared with HEFT, CPOP, and MCP algorithms, especially for large-scale tasks, it can achieve lower task completion time while satisfying load balancing constraints. Future research directions include applying this strategy to real data-intensive applications, optimizing algorithm scalability, and improving and extending model structure transformations by studying decomposition operations to better express and process models.

References

[1] Chen C L P, Zhang Chunyang. Data-intensive applications, challenges, techniques and technologies: a survey on big data [J]. Information Sciences, 2014, 275 (11): 314−347.

[2] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing [J]. International Journal of Computers & Technology, 2010, 53 (4): 50−58.

[3] 宫学庆, 金澈清, 王晓玲, 等. 数据密集型科学与工程: 需求和挑战 [J]. 计算机学报, 2012, 35 (8): 1563−1578. (Gong Xueqing, Jin Cheqing, Wang Xiaoling, et al. Data-Intensive Science and Engineering: Requirements and challenges [J]. Chinese Journal of Computers, 2012, 35 (8): 1563−1578.)

[4] Catalyurek U V, Boman E G, Devine K D, et al. Hypergraph-based dynamic load balancing for adaptive scientific computations [C]// Proc of Parallel and Distributed Processing Symposium. 2007: 1−11.

[5] Zhou D, Huang J, Schölkopf B. Learning with hypergraphs: Clustering, classification, and embedding [C]// Advances in Neural Information Processing Systems. 2006: 1601−1608.

[6] Zhao Han, Liu Xinxin, Li Xiaolin. Hypergraph-based task-bundle scheduling towards efficiency and fairness in heterogeneous distributed systems [C]// Proc of IEEE International Symposium on Parallel & Distributed Processing. 2010: 1−12.

[7] 郭文忠, 陈国龙, 彭少君. 求解 VLSI 电路划分问题的混合粒子群优化算法 [J]. 软件学报, 2011, 22 (5): 833−842. (Guo Wenzhong, Chen Guolong, Peng Shaojun. Hybrid particle swarm optimization algorithm for VLSI circuit partitioning [J]. Journal of Software, 2011, 22 (5): 833−842.)

[8] Ma Yonggang, Tan Guozhen, Wang Wei. A multi-objective hypergraph partitioning model for parallel computing [J]. International Journal of Parallel, Emergent and Distributed Systems, 2012, 27 (4): 337−346.

[9] Biswal P, Lee J R, Rao S. Eigenvalue bounds, spectral partitioning, and metrical deformations via flows [J]. Journal of the ACM, 2010, 57 (3): 13:1−13:23.

[10] Deveci M, Kaya K, Uçar B, et al. Hypergraph partitioning for multiple communication cost metrics: Model and methods [J]. Journal of Parallel & Distributed Computing, 2015, 77 (2015): 69-83.

[11] Devine K D, Boman E G, Heaphy R T, et al. Parallel hypergraph partitioning for scientific computing [C]// Proc of the 20th IEEE International Parallel & Distributed Processing Symposium. 2006: 10.

[12] 孙雪冬, 徐晓飞, 王刚. 基于有向超图的资源约束下企业过程结构优化 [J]. 软件学报, 2006, 17 (1): 59−67. (Sun Xuedong, Xu Xiaofei, Wang Gang. Directed hypergraph based and resource constrained enterprise process structure optimization [J]. Journal of Software, 2006, 17 (1): 59−68.)

[13] Laura L, Nanni U, Temperini M. The organization of large-scale repositories of learning objects with directed hypergraphs [C]// Proc of International Conference on Web-Based Learning. Springer International Publishing. 2014: 23−33.

[14] 肖鹏, 胡志刚, 屈喜龙. 面向数据密集型工作流的能耗感知调度策略 [J]. 通信学报, 2015, 36 (1): 149−158. (Xiao Peng, Hu Zhigang, Qu Xilong. Energy-aware scheduling policy for data-intensive workflow [J]. Journal on Communications, 2015, 36 (1): 149−158.)

[15] 苑迎春, 李小平, 王茜, 等. 基于逆向分层的网格工作流调度算法 [J]. 计算机学报, 2008, 31 (2): 282-290. (Yuan Yingchun, Li Xiaoping, Wang Qian, et al. Bottom level based heuristic for workflow scheduling in grids [J]. Chinese Journal of Computers, 2008, 31 (2): 282−290.)

[16] Ahmad S G, Liew C S, Munir E U, et al. A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems [J]. Journal of Parallel & Distributed Computing, 2016, 87 (C): 80−90.

[17] Lin Bing, Guo Wenzhong, Xiong Naixue, et al. A pretreatment workflow scheduling approach for big data applications in multicloud environments [J]. IEEE Trans on Network & Service Management, 2016, 13 (3): 581-594.

[18] Mon E E, Thein M M, Aung M T. Clustering based on task dependency for data-intensive workflow scheduling optimization [C]// Many-Task Computing on Clouds, Grids, and Supercomputers. 2017: 20-25.

[19] Auffhammer M, Hsiang S M, Schlenker W, et al. Using weather data and climate model output in economic analyses of climate change [J]. Review of Environmental Economics & Policy, 2013, 7 (2): 181−198.

[20] Calheiros R N, Ranjan R, Beloglazov A, et al. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms [J]. Software: Practice and Experience, 2010, 41 (1): 23-50.

[21] Bert A C, source, Norcott. IOzone [M]. 2012.

[22] Juve G, Chervenak A, Deelman E, et al. Characterizing and profiling scientific workflows [J]. Future Generation Computer Systems, 2013, 29 (3): 682−692.

Note: Figure translations are in progress. See original paper for figures.