Communication-aware Fault-tolerant Scheduling Strategy for Precedence Constrained Tasks in Heterogeneous Distributed Systems

Weipeng JING, Yaqiu Liu, Qu WU
School of Information and Computer Engineering, Northeast Forestry University, Harbin 15040, China
doi:10.4156/jdcta.vol5.issue6.30

Abstract

Fault-tolerant scheduling is an important issue for optimal heterogeneous distributed systems because of a wide range of resource failures. Primary-backup approach is a common methodology used for fault tolerance wherein each task has a primary copy and a backup copy on two different processors. For independent tasks, the backup copy can overload with other backup copies on the same processor, as long as their corresponding primary copies are scheduled on different processors. Unfortunately, most of the scheduling algorithms developed on a simple model where communication contention is not taken into account. In this paper we proposed a improve list scheduling algorithm, called heterogeneous communication-aware fault-tolerant scheduling (HCFS). The new approach for computing task priority is proposed which considers the performance difference in heterogeneous systems. Simulation results show that compared with existing scheduling algorithms in the literature, our scheduling algorithm improves the reliability and performance.

Keywords: Fault-tolerant, Primary-backup, Communication Contention, Reliability

1. Introduction

Heterogeneous distributed systems are widely deployed for executing computationally intensive parallel applications with diverse computing needs. Heterogeneous distributed systems are usually composed of diverse sets of resources interconnected with high-speed networks. In order to fully exploit and effectively utilize heterogeneous systems, the flexible and efficiency resource management strategy must be provide to mask the complexity of different processors and link for user. However, hardware and software often can cause failure in an unpredictable way, it is important to provide fault tolerance for it. Hardware redundancy is the main technique to realize fault tolerance, but it is an expensive approach. In order to fully exploit and effectively achieve fault tolerance, some fault-tolerant scheduling algorithms have been proposed in heterogeneous distributed systems which based on directed acyclic graph (DAG), where a node represents a task and an edge is the communication between two tasks. Task scheduling has been proposed in literature [1, 2, 3, 4]. The task duplication method is taken which is based on space redundancy. The multiple copies of current task are mapped on different processors, which are executed in parallel to tolerate a number of failures [5, 6]. However, it will produce massive redundancy task to be executed. On the other hand, the primary-backup scheme plays an important role in achieving fault tolerance [7, 8, 9, 10]. The task scheduling model is proposed to maximize the system’s reliability in heterogeneous systems.

In addition, all these task scheduling algorithms are usually studied by using macro-dataflow model. In other words, the communication network is assumed to be contention-free, which of course is not realistic as soon as the processor number exceeds a few units. Recently, a few communication contention algorithms were proposed which consider network contention [11, 12] or end-point contention [13, 14]. The new system model for task scheduling is capable of capturing both end-point and network contention [15]. All previous contention aware models were developed for minimizing latency on realistic parallel heterogeneous systems; they do not deal with fault-tolerance. In [16] contention awareness and fault-tolerant scheduling was proposed. Unfortunately, the active replication (N-modular redundancy) is based on space redundancy.

All pervious fault-tolerant algorithms do not accurately analyze the start time of primary and backup tasks. So the makespan (or the schedule length) of DAG is not accurate. In this paper, we propose a
novel scheduling algorithm based on both list-based approach and primary-backup techniques for distributed heterogeneous distributed systems. In order to reduce replication consumption, the primary-backup duplication of tasks in our algorithm is optimized for all immediate parent tasks. We use passive and overlapping backup and analyze the processor and link reliability and derive a reliability cost formula. The rest of the article is organized as follows: in the next section, we define the task scheduling problem model. The heterogeneous communication-aware fault-tolerant scheduling (HCFS) algorithm is proposed in Section 3. In Section 4, the simulation experimental results are presented and analyzed. The article concludes with Section 5.

2. System Model

An efficient scheduling of application tasks is critical in order to achieve high performance in parallel and distributed systems. The objective of scheduling is to find a mapping of the tasks onto the processors, and to order the execution of the tasks.

2.1. Parallel Task and Heterogeneous System Model

A real-time task can be modeled by Directed Acyclic Graph (DAG) $G = (V, E)$, where $V$ is the set of nodes corresponding to the tasks, and $E$ is the set of edges corresponding to the precedence relations between the tasks. The task $t_i$ in $G$ includes two parts, one is the scheduled tasks $S$ and the other is unscheduled tasks $U$. $t_i$ is the $i$-th task and $n$ is the total number of tasks. A set of weighted and directed edges $E$ represent communication among tasks. $v(t_i, t_j) \in E$ indicates the volume of message that $t_i$ needs to transmit to $t_j$. The link between tasks $t_i$ and $t_j$ is denoted by $\lambda_{ij}$.

A heterogeneous system is considered to consist of finite heterogeneous processors set $P = \{P_1, P_2, ..., P_m\}$. $P_i$ is the $i$-th processor and $m$ is the total number of processors. These processors are assumed to be fully connected. The computational heterogeneity of tasks is modeled by a function $\epsilon: V \times P \rightarrow R^+$, which represents the execution time of each task on each processor in the system. $\epsilon(t_i, P_k) (1 \leq k \leq m, 1 \leq i \leq n)$ denotes the execution time of $t_i$ on $P_k$. The heterogeneity in terms of communications is expressed by $W(t_i, t_j) = v(t_i, t_j) \times d(P_i, P_j)$, where are mapped on the same processor. $d(P_i, P_j) = 0$. The communication link between processor $P_i$ and $P_h$ is denoted by $L_{i,h}$.

2.2. Fault Model

The processor of $P$ are assumed to be fail-silent. All the failures are transient and we assume that their maximal duration is such that a fault isolation mechanism is provided, so that a faulty processor cannot cause incorrect behaviors in non-faulty processors.

To concentrate on our concerned problems, we make the following assumptions:

(1) Failure occurrences are supposed to be statistically independent events, and the occurrence of a failure on a processor $P_i$ follows a Poisson’s law with constant parameter $\lambda_i$. Thereby, the probability that task $t_i$ is mapped on processor $P_i$ is $R(t_i, P_i) = e^{-\lambda_i \times \epsilon(t_i, P_i)}$.

(2) The fail-stop manner is used in processors failure, which means a processor is either operational or cease functioning;

(3) The failure of a processor is detected by the remaining processors within the closest completion time of a task scheduled on the faulty processor and the detecting time is ignored.
2.3. Adaptation to the Communication Contention Model

In order to adapt communication contention model, we take constraints related to communication resources into account. If a communication link has been scheduled, the ready time of the link is the time at which this last communication terminates. So the time complexity of the algorithm is drastically reduced. And we must need to research the whole time span of each link to find the earliest time slot. $R(\ell)$ is defined as the ready time of a communication link $\ell$. So the start time and finish time of communication link $l$ which the data is sent from $t_i$ to $t_j$ is defined as $t_{i,j}$ and $t_{i,j}$. In the following, we formalize all communication constraints.

For any two communication $m$ and $m'$ scheduled on link $\ell$. It is to say that any two communications do not overlap on a link.

$$L_{i,k}^{f,m} \leq L_{i,k}^{f,m'} \vee L_{i,k}^{f,m'} \leq L_{i,k}^{f,m}$$

(1)

For any two communication $m$ and $m'$ sent from a given processor $P_k$ respectively to two processor $P_k$ and $P'$. The sending message constraint is defined as

$$L_{i,k}^{f,m} \leq L_{i,k}^{f,m'} \vee L_{i,k}^{f,m'} \leq L_{i,k}^{f,m}$$

(2)

For any two communications $m$ and $m'$ sent from processors $P_k$ and $P'$ to the same processor $P_k$. The receiving message constraint is defined as

$$L_{i,k}^{f,m} \leq L_{i,k}^{s,m'} \vee L_{i,k}^{s,m'} \leq L_{i,k}^{s,m}$$

(3)

It is to say that only one interface can be used.

At lst let $[A, B]$ is an idle time interval which no task is executed on processor $P$, a free task $t_i$ can be scheduled on $P$ within $[A, B]$ only if

$$\max \{A, DRT(t_i, P)\} + \epsilon(t_i, P) \leq B$$

(4)

$DRT(t_i, P)$ is called the data ready time. Hence the condition allows task $t_i$ to be scheduled within already scheduled tasks.

3. Communication-Aware Fault-Tolerant Scheduling

The scheduling problem is to find a schedule with minimal length. As this problem is NP-hard, many heuristics have been proposed for its solution. A heuristic must schedule a node on a processor so that it fulfills all resource and precedence constraints. The best known scheduling heuristic is list scheduling.

In this section, we will explore the minimum completion time of primary and backup tasks in heterogeneous system. We schedule backup tasks and their primaries overload on different processors in this paper. Edge scheduling, thanks to its strong similarity with task scheduling, has the potential to accurately reflect contention in communication. Therefore, the here adopted approach to contention awareness is based on the edge scheduling concept. And we discussed these edge scheduling constraints and found makespan of primary and backup task.
3.1. Task Priorities

First we must consider allocating primary and backup copies to processors which assign task copies by its priorities. This approach simplified the algorithm since only lower priority tasks are assigned later to the same processor hence time intervals for already assigned tasks will remain unchanged. In this paper, tasks are ordered by their scheduling priorities based on ranking \[17\]. The rank of task \( t_i \) is recursively defined as

\[
\text{rank}(t_i) = \varphi(t_i, P) + \max_{j \in \text{succ}(s_i)} \{ W(t_i, t_j) + \text{rank}(t_j) \}
\]

(5)

Where \( \text{succ}(i) \) is the set of immediate successors of task \( t_i \). The rank is computed recursively by traversing the task graph. For the value of exit task \( t_{\text{exit}} \) is equal to \( \text{rank}(t_{\text{exit}}) = \varphi(t_{\text{exit}}, P) \). The rank of backup task \( t_i \) is same as its primary task.

3.2. Communication-Aware Scheduling

The primary-backup task duplication under the contention model changes significantly. Under the contention model, it must be strictly defined from where a communication is sent if there are several instances of a sending task. Let \( t_i \in \text{pred}(j) \) denote a task on processor \( P_k \). \( t_i^{p,f} \) is the time at which processor \( P_k \) has finished the processing primary of task \( t_i \). Let \( R(P_k) \) denote the sending free time of primary task \( t_i \) on processor \( P_k \), the time at which all sending communications already scheduled on processor \( P_k \) are finished. \( R(P_k) \) is backup one. \( R(\ell) \) and \( R(\ell') \) are defined as the ready time of a communication link \( \ell \) which take the message of primary and backup task \( t_i \) to task \( t_j \). So the start time and finish time of communication link \( \ell \) which the data is sent from \( t_i \) to \( t_j \) is defined by the following equations

\[
\ell_{i,j} = \max(t_i^{p,f}, t_i^{b,f}, R(P_k), R(P_k'), R(\ell), R(\ell'))
\]

(6)

\[
\ell_{i,j}^f = \ell_{i,j} + W(t_i, t_j)
\]

(7)

Thus, communication \( \ell_{i,j} \) is constrained by both \( R(\ell) \) and the finish time of its source task \( t_i \) on \( P_k \). It can start as soon as the processing of the task is finished only if we have \( R(P_k) \leq t_i^{p,f} \) and \( R(\ell) \leq t_i^{p,f} \).

The start time of task \( t_j \) on processor \( P_k \) is constrained by communication incoming from its predecessors that are assigned on other processors. Let \( \text{DRT}(t_j, P_k) \) be the time when communication arrives on processor \( P_k \). And, hence \( \text{DRT}(t_j, P_k) = \max_{i \in \text{pred}(t_j)} \{ \ell_{i,j} \} \). The ready time of processor \( P_k \) denote \( r(P_k) \). The start time of primary \( t_j \) on processor \( P_k \) is defined as follows:

\[
\ell_{j}^{p,f} = \max(\max\{\text{DRT}(t_j, P_k)\}, r(P_k))
\]

(8)
So the earliest start time $EST(t_j^p)$ of the primary copy of $t_j$ is defined as:

$$EST(t_j^p, P_1) \geq t_j^p$$

(9)

If $t_j$ can be scheduled with no additional constraints. Then we have the finish time $EFT$ which is computed as follows

$$EFT(t_j^p, P_k) \geq EST(t_j^p, P_k) + e^p(t_j, P_k)$$

(8)

Then, we discuss how to schedule backup task $t_j$, which may overload with other primary task considering all its predecessors. However, a key question to be found is that primary of $t_j$ may start before backup of $u$. If $t_j^p < \max_{u \in \text{pred}(t_j)} t_u^b$, then the backup of $t_j$ must be satisfied with the following two conditions.

$$\text{condition 1: } t_j^b > \max_{u \in \text{pred}(t_j)} t_u^b$$

$$\text{condition 2: } P_j^b \neq P_u$$

(10)

Then it is to say backup of task $t_j^b$ can only start after backup of task $t_u^b$ finishes and must not be scheduled on the processor $P_u$. We denote the set of its direct predecessors as set $u'$. For example, as shown in Fig. 1, we consider scheduling task $t_1$ with its predecessor’s tasks $t_m$ and $t_n$. we can see that $t_j^b$ cannot be scheduled on processor P1 or on processors P2 and P4 before $\max_{u \in \text{pred}(t_1)} t_u^b$. Where primaries of $t_m$ and $t_n$ are located. Otherwise, when processor 1 or 2 fails, backup of $t_j^b$ also fails while primary of $t_j^b$ cannot receive result. This task cannot be completed. Therefore, backup of $t_j^b$ can only be scheduled on processor 3 after backup of $t_{u'}^b$.

Fig 1. Scheduling task considering predecessors tasks
It is easy to say that backup of task $t^b_j$ can only start when backup of task $t^b_{i'}$ finished, more important they must not be scheduled on the processor where primary of $t^p_{i'}$ is located. $u'$ denote the set of its direct predecessors. The last start time of backup of task $t^b_j$ is defined as

$$t^b_j > \max_{u \in \text{pred}} \{t^b_u, t^b_{u'}\}$$

(11)

So the earliest start time and the finish time for the backup copy of $t^b_j$ are

$$EST(t^b_j, P_a) \geq t^b_j$$

(12)

$$EFT(t^b_j, P_a) \geq EST(t^b_j, P_a) + e^b_j(t_j, P_a)$$

(13)

From above we attempts to derive the time when the task $t^b_j$ can start so that it can receive all message from its predecessors even when predecessors fault happens. After all tasks in a graph are scheduled, which length will be the finish time of the exit task $t^e_{exit}$, thus the schedule length of the task graph is defined as $\text{makespan} = EFT(t^e_{exit})$.

### 3.3. Complexity Analyze

We now analyze the complexity of this algorithm. We know the time complexity of sorting algorithm is $O(M \log N)$, the main computational cost of HCFS is spent in the while loop (Line 5 to 11), the loop is executed $M$ times. Line 6 costs is $O(\log N)$, Line 7 to 10 is compute the reliability which include processor and link and the makespan, so the costs is $O(eN)$, while computing $EFT(t^f_j, P_a)$, $EFT(t^b_j, P_a)$. Thus the total cost of HCFS is $O(eN + M \log N)$

---

**Algorithm HCFS**

1: $P = \{P_1, P_2, \ldots, P_M\}$; (*set of processors*);
2: Compute $\text{rank}(t_i)$ for each task $t_i$ in $U$;
3: $S = \emptyset, U = V$;
4: List of free tasks;
5: while $U \neq \emptyset$ do;
6: $t_i < - H(t_i)$; (*select task with highest priority*)
7: Compute $EFT(t^f_i, P_a)$
8: Compute $EFT(t^b_i, P_a)$
9: Schedule $t_i$ to the corresponding processor
10: Put $t_i$ in $S$
11: Update the priority in $U$
12: end while

---

**Algorithm 1.** The pseudo-code for HCFS algorithm
4. Experiments

The experiments are made to evaluate the performance of HCFS algorithm. The parameters which are used in the experiments are chosen as the following. And we take the experimental performance comparison between HCFS and HEFT. The makespan and reliability are the two main metrics to test performance. All experimental data presented in this paper are the mean of 10 experimental results.

A Poisson distribution is used to generate the task arrivals and the average task inter-arrive time is 10s. The number of computation processors in the distributed heterogeneous system used in the simulation experiments is fixed at $10 \leq M \leq 50$. The number of tasks is fixed at $100 \leq N \leq 1000$.

4.1. Performance Comparison between HCFS and HEFT

This experiment evaluates performance in terms of reliability cost and makespan between HCFS and HEFT. The workload consists of sets of independent real-time tasks running on a homogeneous system. We can see from Fig.2 and Fig.3 the HCFS algorithm is more effectively able to deal with the increase in communication cost compared HEFT algorithms. This is because allocating many copies of tasks will improve the reliability and short the communication delay.

4.2. Scheduling Length Ratio

In this section, we compare the makespan between HCFS and HEFT. Since a large set of application tasks with different properties is used in the heterogeneous system. It is necessary to normalize the schedule length to the lower bound, which is called the Scheduling length ratio (SLR). It is defined as:

$$ SLR = \frac{\text{makespan}}{\sum_{i \in U} \min_{t \in P} c(t_i, P)} $$

We utilized the average SLR values to assess the performance of HCFS. We can see from the Fig.4, the average SLR of HCFS is shorter than HEFT. This is because that the start time of the HCFS is much earlier than HEFT.

\[\text{Fig 2. The reliability comparison between HCFS and HEFT}\]

\[\text{Fig 3. The Reliability cost between HCFS and HEFT}\]
Communication-aware Fault-tolerant Scheduling Strategy for Precedence Constrained Tasks in Heterogeneous Distributed Systems
Weipeng JING, Yaqiu Liu, Qu WU

5. Conclusion

In this paper, we have addressed the problem of fault-tolerant list scheduling in heterogeneous computing system which takes into communication contention account. The scheduling algorithm which is discussed in this paper achieves minimum makespan and maximum reliability. Simulation studies show that the solution quality and the time complexity of our algorithm make it a viable choice for the compile-time scheduling.

6. Acknowledgement

This work is supported by 948 project (2010-4-05), the Fundamental Research Funds for the Central Universities (DL09BB04) and the Fundamental Research Funds for the Central Universities (DL09DB03), Scientific Research Fund of Heilongjiang Provincial Education Department (11553022)

7. References

[10] Dost Muhammad Khan, Nawaz Mohamudally, "A Multiagent System (MAS) for the Generation of Initial Centroids for kmeans Clustering Data Mining Algorithm based on Actual Sample Datapoints", 

Fig 4. The average SLR between HCFS and HEFT

2

1.6

1.65

1.7

1.75

1.8

1.85

1.9

1.95

2

Number of tasks

AVergae SLR

HCFS

HEFT
Communication-aware Fault-tolerant Scheduling Strategy for Precedence Constrained Tasks in Heterogeneous Distributed Systems
Weipeng JING, Yaqiu Liu, Qu WU