Enhancing Parallel Data Mining Performance on a Large Cluster by Using UCE Scheduling

Nunnapus Benjamas and Putchong Uthayopas
Department of Computer Engineering, Kasetsart University, Bangkok 10900, Thailand
{g4885043,pu}@ku.ac.th

Abstract
In this paper, we propose an algorithm called Unified Communication and Execution Scheduling (UCE) that combines the execution and communication scheduling for parallel data mining application together. This algorithm enables a better utilization of hardware and interconnection in a multicore cluster system for the data mining application. The idea is to choose a proper task execution sequence combined with a communication scheduling that avoids the communication conflict in the interconnection network switch. The simulation results show that a substantial performance improvement can be obtained especially with a multicore cluster system.

1. Introduction
Currently, Human society is striving to understand more about the relationship among complex data collected from everywhere. This is substantially important for many domains such as inferring patterns from web page access logs, bioinformatics, and the analysis of medical, scientific, and commercial data [1]. Thus, a fast and scalable data mining technique becomes increasingly more important. Faster processing speed enables users to experiment with various approaches and processes a much larger data set. Therefore, many researchers have worked on improving the large-scale parallel and distributed data mining algorithms on scalable parallel computing systems ([2], [3], [4], [5], [6], [7], [8], [10], [11], [12], [14], [15], [16], [18]). These include the shared memory multiprocessor system [9] and distributed memory multicomputer system [13], [17], [19], [37] or cluster. Since the parallel data mining application typically generates a lot of communication messages, the optimization of the communications structure by employing special characteristics of the computing system can result in a great performance enhancement.

In this paper, we propose a scheduling strategy called Unified Communication and Execution Scheduling (UCE). In this approach, the communication and execution of the subtasks of parallel data mining will be linked together to improve the performance. The parallel FI-growth data mining [20] is used to demonstrate the proposed concept. Using simulation, UCE scheduling is shown to give a substantially improved performance for parallel FI-Growth application.

The rest of this paper is organized as follows. Section 2 describes related researches in this area. Section 3 presents the system model followed by Section 4 that discusses the design of a right scheduling strategy. Section 5 shows our experimental evaluation. Finally, Section 6 presents the conclusion and future work.

2. Literature Review
The Apriori algorithm [21], [22] has emerged as one of the best classical algorithm for the association rule mining since 1994. The used approach is based on the “generate-and-test” paradigm. To enhance the performance, Park et al. [17] proposed the DHP algorithm (standing for direct hashing and pruning) which is an extension of the Apriori algorithm. This hash-based technique can reduce number of generated candidate itemsets, so that it affects less computational cost in later iterations. Han et al. [23] presented a new algorithm for mining association rules called the FP-growth algorithm. This algorithm finds a complete set of frequent itemsets using a Frequent Pattern tree (FP-tree) structure to avoid candidate itemsets generation. Recently, Amphawan and Surarerks [24] presented the Frequent Item growth (FI-growth) algorithm, for creating a new FP-tree called a Frequent Item tree (FI-tree). They have shown that the complete set of frequent itemsets can be generated using a single tree. Anyway, as the size of the data grows, use of parallel data mining algorithms becomes necessary to obtain the result in a reasonable time.
Several parallel data mining algorithms have been designed for association rule mining [21], [25], [26]. Many of them employ the generate-and-test approach [12], [17], [27]. For example, Agrawal and Shafer [12] proposed three parallel approaches for mining association rules: count distribution, data distribution, and candidate distribution. There are some works that use pattern growth approach [2], [9], [13], [28] based on FP-tree [23]. For the parallel version of FP-growth algorithm, the existing parallel attempt on cluster of shared-nothing environment was reported in [29]. The PDM algorithm [30] is a parallel version of the DHP, i.e., each processor determines the global supports of 1-itemsets through an all-to-all broadcast and then approximates the counts for 2-itemsets with a hash table. Zaki [25] also discussed several parallel association rule-mining algorithms on various parallelization schemas. Recently, efficiently partitioning database methods for optimal load-balance and minimum communication overheads have also been reported in [9], [13], [33], [34], respectively.

Hardware platform issues are also addressed in the parallel implementation of association rule mining, such as shared memory [9], [32] and distributed memory [17], [13], [28], [35]. Most of the works address how an association rule-mining algorithm will perform on an ideal distributed system. Nevertheless, parallel data mining applications usually generates a lot of communication messages. Thus, a proper optimization of these communications on an infrastructure has a potential to substantially increase the performance of a parallel data mining application.

3. Parallel FI-Growth Application

In order to understand the proposed approach, the Parallel FI-growth [20] used for the study is explained. This algorithm is an algorithm for parallel data mining that parallelizes the association rule mining process. The algorithm employs a data parallelism technique on a multicore cluster (see the workflow in [20]). The workflow consists of 3 phases, which are preprocessing phase, FI-tree construction phase, and mining phase.

In the preprocessing phase, we first partition the transaction database into several portions, and distribute them to different processors for computation. In the FI-tree construction phase, each processor independently constructs its own local FI-tree structure and discovers corresponding frequent itemsets. However, all processors need to perform a one-time synchronization to exchange their sub-trees before the last two steps in the mining phase.

3.1. Application model

In this paper, the parallel application is modeled by using a Directed acyclic graph (DAG), which is used as an input to the scheduling simulator.

![Figure 1. The parallel FI-growth task graph](image)
In Figure 1, let \( T = \{ T_0, T_1, \ldots, T_n \} \) denote a set of tasks to be executed, a directed acyclic graph can be defined as a graph \( G = (V, E) \), where \( V \) is a set of nodes \( \{ v_1, v_2, \ldots, v_n \} \) and \( E \) is a set of directed edges \( \{ e_{ij} \} \). The edge \( e_{ij} \) in the DAG connecting nodes \( v_i \) and \( v_j \) represents the communication and precedence constraints among tasks. Let the weight of an edge \( W(e_{ij}) \) denote the communication cost between two tasks. A node \( v_i \) in the DAG corresponds to task \( T_i \). Each node of the task graph has a weight \( W(v_i) \) that represents the computing cost.

In this application model, the execution task is assumed to be as indivisible unit of computation, and each processor executes one task at a time. Upon receiving all the input data, the task can start the execution. For communication contention, a task cannot receive data from all its predecessors simultaneously, but it receives data sequentially from the same communication channel.

### 3.2. System model

In this paper, a multicore cluster system is used as the target architecture. A multicore cluster system is a set machine \( M = \{ m_1, m_2, \ldots, m_{|M|} \} \) of \( |M| \) machines, and each machine \( m_i \) includes a set processor \( P_i = \{ p_1, p_2, \ldots, p_{|P_i|} \} \) of \( |P_i| \) processors. Each processor \( p_j \) of machine \( m_i \) has multiple core \( C_i = \{ c_1, c_2, \ldots, c_{|C_j|} \} \) of \( |C_j| \) cores. The total number of processors and the total number of cores are \( P_{\text{total}} = \sum_{i=1}^{M} |P_i| \) and \( C_{\text{total}} = \sum_{i=1}^{M} \sum_{j=1}^{|P_i|} |C_j| \) respectively.

In Figure 2, the multicore cluster system is a tree-structure with cores (c) as leaves, processors (p) as intermediate nodes being a parent for cores, machines (m) as intermediate nodes combining processors and the entire machine or system (S) as root node. Let \( S = (V, E, W_S, W_E) \), \( V = \{ v_1, v_2, \ldots, v_n \} \) be a set of nodes, and \( E \) be a set of undirected edges. \( V \) represents the set of cores in the system, and an edge \( e_{ij} \in E \) represents communication link between core \( v_i \) and \( v_j \). \( W_S(v_i) \) and \( W_E(e_{ij}) \) are the computational cost of core \( v_i \) and the communication cost between core \( v_i \) and \( v_j \), respectively.

![Figure 2. System model](image)

In general, a multi-core cluster system is connected together, using a hierarchy of communication networks. First, an **inter-core communication** uses the shared-cache between cores on the same processor. Second, an **inter-processor communication** uses shared-memory between cores on the different processor. Finally, an **inter-machine communication** has to go through interconnection network that links these machines together. In this work, we assume that the bandwidth between cores is 20 GBps, bandwidth between processors is 10 GBps, and bandwidth between machines is 1 GBps.

It is assumed that the number of processors and the number of tasks are static and known beforehand. In this paper, we use an all-to-all personalized communication to illustrate that a proper communication scheduling of FI-growth data mining can substantially increase the speed of the application. In general, interconnection network in a cluster usually consists of a switch fabric and a network links, which connect computation node with a certain bandwidth such as 1 Gbps. When more than one communication session is initiated to a single target node, bandwidth linking the switch and node will be shared. The bandwidth is then reduced, and more latency is inserted into the computation. An example is given in Figure 3.
In Figure 3a, it is a fully connected network. Thus, every machine \( m \) has its own link with every other machine via switch (SW). Figure 3b, machine \( m_2 \) and machine \( m_6 \) need to send messages to machine \( m_3 \) at the same time. To avoid this conflict, we proposed that the communication should be scheduled. The benefit of the proposed strategy is that the communication between a pair of node is contention free, as shown in Figure 3c. (Communication scheduling is presented in greater detail in Section IV.) To express communication contentions, we assume that only a limited number of communications can pass from the processor into the network and from the network into the processor at one communication at the same time.

In our previous work [36], we proposed a strategy called Execution Only Scheduling (EO) to improve the application performance. In EO method, the scheduling is divided into two steps; ordering of the tasks, and the mapping of the tasks onto processing units. The detail is given below.

**Ordering step**: The task graph represented as a DAG is assigned a priority. We use smallest-numbered available task first or Left-to-Right (L-R) to determine the priorities of nodes. For example, a scheduling list from task graph in Figure 1 is \( T_0, T_1, T_2, T_3, T_5, T_7, T_9, T_10, T_11, T_12, T_13, T_14, T_{15}, T_{16}, T_{17}, T_{19}, T_{20}, T_{21}, T_{22}, T_{23}, T_{24}, \) and \( T_{25} \).

**Mapping step**: The mapping considers two factors, the largest data transfer size and the earliest start-time. First, task will be mapped onto the same node as its parents that generate the largest data transfer size to that task. If more than one parent nodes exist with the same largest transfer size, the processing unit with earliest start-time first is selected.

In this work, we propose a strategy based on the previous that can even give a better performance than the last scheme. This is explained in the following section.

### 4. Designing the Scheduling Strategies

In order to further enhance the performance, we propose an approach called **Unified Communication and Execution Scheduling (UCE)**. The concept is to combine both computation and communication scheduling together to achieved a much better performance.

In UCE scheduling, the execution is decomposed into two steps, the nodes partitioning and communication scheduling. These two steps execute alternately. For each step, the execution will proceed as follows:

**Partitioning step**: A set of \( n \) compute node is divided into two balanced partitions and then recursively divided the two partitions until the number of node of each partition is one. The total number of partitioning step is \( \log_2 n \).

**Communication step**: In each partitioning step, all nodes in each partition will be scheduled to send/receive message to/from all nodes within another partition. The number of communication steps of each partitioning step will be equal to the number of nodes in the divided partition. The total number of communication steps is \( n-1 \) for all nodes. Therefore, the total number of the exchange of data between all nodes is \( (n-1)n \). In this approach, the communication is scheduled with computation so that the communication can take place in a step that avoids the conflict in bandwidth sharing. Thus, the communication bandwidth between nodes can be fully utilized. Hence, the faster communication can take place and a lower total execution time is achieved.
To help clarify the proposed strategy, an example of partitioning step and communication step of bandwidth scheduling strategy of eight nodes \((n=8)\) is illustrated in Figure 4. In this case, the total number of partitioning step of this example is \(\log_2 8 = 3\). The total number of communication step is \(8-1=7\). Accordingly, the total number of exchange of data between all nodes is \((8-1)\times 8 = 56\). The example of results of bandwidth scheduling strategy of eight nodes is given in Fig 5.

**Figure 4.** The example of partitioning step and communication step of bandwidth scheduling strategy of eight nodes.
Figure 5. The example of results of bandwidth scheduling strategy of eight nodes

5. Experimental Evaluation

In this work, a database of 60 million transaction database is mined by using the parallel FI-growth program [20] to generate the test task graph. We utilized the standard “IBM synthetic data generator” [31] to synthesize a transaction database. We used 1000 unique items to create 60 million records, which have average transaction length of 10. The communication volume associated with each edge varies from 96 bytes to 768.8 Kbytes. After the test graph is generated, we can run a scheduling simulator on this task graph using different scheduling strategies. All the scheduling strategies in our simulator have been written in Java programming language and running under Microsoft Windows. To evaluate these scheduling strategies, we ran all of the considered strategies on variation architecture as shown in Table 1.

<table>
<thead>
<tr>
<th>Multicore Single system</th>
<th>CPU</th>
<th># of CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>Dual-core</td>
<td>1</td>
</tr>
<tr>
<td>M2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>Quad-core</td>
<td>1</td>
</tr>
<tr>
<td>M5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>M6</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>M7</td>
<td>Hexa-core</td>
<td>1</td>
</tr>
<tr>
<td>M8</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>M9</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>M10</td>
<td>Octo-core</td>
<td>1</td>
</tr>
<tr>
<td>M11</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>M12</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multicore Cluster system</th>
<th>CPU</th>
<th># of machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Dual-core</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>C6</td>
<td>Quad-core</td>
<td>2</td>
</tr>
<tr>
<td>C7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>C9</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>C10</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>C11</td>
<td>Hexa-core</td>
<td>2</td>
</tr>
<tr>
<td>C12</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>C13</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>C14</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>C15</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>C16</td>
<td>Octo-core</td>
<td>2</td>
</tr>
<tr>
<td>C17</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>C18</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>C19</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>C20</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>
Figure 6a, 6b, 6c, and 6d illustrate the run times for UCE and EO scheduling strategies on dual-core, quad-core, hexa-core, and octo-core single system, respectively. The results clearly show that the UCE scheduling strategy gives a lower execution time than the EO scheduling strategy. In the best case, the runtime is reduced as much as 14.70%.

Figure 7a, 7b, 7c, and 7d show the run times for UCE and EO scheduling strategies on dual-core, quad-core, hexa-core, and octo-core multicore cluster system, respectively. The results show that run-time drops rapidly as the number of core increase until 32 cores, and then run-time drops more slowly as additional cores are used. This is caused by the additional communication overhead incurred. For all variation multicore cluster systems, the largest runtime is 51.45%. We also found that the decreased run time of UCE and EO on multicore cluster system was more than multicore single system in case of equal number of cores. This is the result of a better utilization of interconnection network used to build the system.
6. Conclusions

In this paper, a new strategy for parallel-FI growth data mining called UCE (Unified Communication and Execution) is proposed. By combining the efficient execution and communication scheduling together, a substantial performance improvement can be achieved. Currently, we are working on how to extract performance from large hybrid architecture such as GPU based cluster. In addition, we are looking forward to applying this approach to other algorithms. We hope that this work can lead to the development of data mining algorithms that can handle a massive scale data faster and lead us to discover more knowledge that is useful for us.

7. References

Enhancing Parallel Data Mining Performance on a Large Cluster by Using UCE Scheduling
Nunnapus Benjamas and Putchong Uthayopas