THESIS APPROVAL

GRADUATE SCHOOL, KASETSART UNIVERSITY

Doctor of Engineering (Computer Engineering)
DEGREE

Computer Engineering
FIELD

Computer Engineering
DEPARTMENT

TITLE: High Performance Parallel Association Rule Mining on Multi-Core Cluster

NAME: Ms. Nunnapus Benjamas

THIS THESIS HAS BEEN ACCEPTED BY

THESIS ADVISOR
(Assistant Professor Putchong Uthayopas, Ph.D.)

THESIS CO-ADVISOR
(Assistant Professor Arnon Rungsawang, Ph.D.)

DEPARTMENT HEAD
(Assistant Professor Putchong Uthayopas, Ph.D.)

APPROVED BY THE GRADUATE SCHOOL ON _____

DEAN
(Associate Professor Gunjana Theeragool, D.Agr.)

THESIS

HIGH PERFORMANCE PARALLEL ASSOCIATION RULE MINING ON MULTI-CORE CLUSTER

NUNNAPUS BENJAMAS

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of
Doctor of Engineering (Computer Engineering)
Graduate School, Kasetsart University
2012

Nunnapus Benjamas 2012: High Performance Parallel Association Rule Mining on Multi-Core Cluster. Doctor of Engineering (Computer Engineering), Major Field: Computer Engineering, Department of Computer Engineering. Thesis Advisor: Assistant Professor Putchong Uthayopas, Ph.D. 53 pages.

Association rule mining has been deployed in many decision support systems. To extract significant patterns and generates some interesting rules from very large dataset. These rules can help users make better decisions and bring enormous benefits to the society. Faster processing speed enables users to experiment with various approaches and processes a much larger dataset. Thus, a fast and scalable data mining technique is very important. There are many challenges in accelerating a large scale data mining such as proper data decomposition and I/O minimization, work load balancing and communication minimization.

This thesis proposes various strategies to speed up parallel association rule mining. First, improving the I/O performance by selecting a proper data distribution and I/O scheduling on the system. Second, propose the scheduling strategy to increase the efficiency of the task execution. Finally, propose an effective scheduling of communication message that take into consideration the message size and interconnection network utilization. These proposed strategies are combined into an approach called Unified I/O, Communication and Execution Scheduling (U-ICE).

From the extreme evaluation using simulation, the U-ICE approach can great improve the speed of a parallel association rule mining on a large dataset. This thesis allows one to efficiently process a massive dataset which has very board applicability in many areas.

_____ _____ ___ / ___ / ___

Student's signature Thesis Advisor's signature

ACKNOWLEDGEMENTS

To accomplish all of this work there are many people who provide supports and reviews. First of all, I would like to gratefully thank my advisor, Assistant Professor Dr. Putchong Uthayopas for his grateful advices and valuable knowledge.

I would like to thank my thesis committees, Assistant Professor Dr. Arnon Rungsawang, for their time and effort. In addition, I would also like to extend my gratitude toward many lecturers in the department of Computer Engineering for their contribution directly or indirectly to this research.

The friendly environment inside this department is one of the key that bring me to the final phase. I would like to thank you to my colleagues in High Performance Computing and Networking Center (HPCNC) and to thank every people in department of Computer Engineering for their many helps during this time.

Special thank you many lecturers in the department of Computer Science, Faculty of Science, KhonKaen University, for their many helps and support. Above all, I would like to thank my parent and my brother for their kindly support and love which make me strength and reach to this moment.

Nunnapus Benjamas

September 2012

TABLE OF CONTENTS

Page

TABLE OF CONTENTS	i
LIST OF TABLES	ii
LIST OF FIGURES	iii
INTRODUCTION	1
OBJECTIVES	3
LITERATURE REVIEW	4
MATERIALS AND METHODS	21
Materials	21
Methods	21
RESULTS AND DISCUSSION	34
Results	34
Discussion	43
CONCLUSION	45
LITERATURE CITED	48
CURRICULUM VITAE	53

LIST OF TABLES

Table		Page
1	Example of a transaction database	6
2	Classification of association rule mining algorithms	18
3	Possible data distribution strategies	26
4	Possible scheduling strategies	29
5	System Variation Architecture	34
6	Multi-core variation architecture	35
7	Multi-core single system variation architecture	38
8	Multi-core cluster system variation architecture	38

LIST OF FIGURES

Figure		Page
1	The initial FI-Tree structure: the header table (left) and the items-tree (right)	7
2	Result of combining operation applied to two sub-trees tree(c)	8
3	The final items-tree derived from the transaction database in Table 1	8
4	The parallel FI-growth work flow	9
5	The initial FI-Tree structures residing at the processor p_0 (left) and processor p_1 (right)	12
6	The evolution of FI-Tree structures after the branching step	12
7	Illustration of typical multi-core clusters	14
8	The trends in term of percentage of the I/O time (I/O), computing time (Comp), and communication time (Comm).	15
9	Association rule mining review	20
10	The parallel FI-growth task graph	22
11	A multi-core cluster system	23
12	System model	24
13	Three data distributions	26
14	The pseudo code of I/O scheduling algorithm	27
15	(a) Network model, (b) More than one communication sessions are initiated to a single target node. (c) The communication between a pair of node is contention free.	30
16	The example of partitioning step and communication step of bandwidth scheduling strategy of eight nodes.	32
17	The example of results of bandwidth scheduling strategy of eight nodes	33
18	I/O time of vary data distribution of experimental setup no. 1, 2, 3, and 4	35
19	Run time using (a) 8 cores, (b) 16 cores, (c) 32 cores, and (d) 64 cores on variation architectures	36
20	Workload of parallel FI-growth of the scheduling variation strategies on 32 cores (a) strategy no. 1, 2, and 3, (b) strategy no. 4, 5, and 6, and (c) strategy no. 7, 8, and 9	37

LIST OF FIGURES (Continued)

Figure		Page
21	Runtime of parallel FI-growth using the U-ICE and EO scheduling strategies on (a) Dual-core, (b) Quad-core, (c) Hexa-core, (d) Octo-cores on multi-core single system variation architectures	39
22	Speedup of parallel FI-growth using the U-ICE and EO scheduling strategies on (a) Dual-core, (b) Quad-core, (c) Hexa-core, (d) Octo-cores on multi-core single system variation architectures	40
23	Runtime of parallel FI-growth using the U-ICE and EO scheduling strategies on (a) Dual-core, (b) Quad-core, (c) Hexa-core, (d) Octo-core on multi-core cluster variation architectures	41
24	Speedup of parallel FI-growth using the U-ICE and EO scheduling strategies on (a) Dual-core, (b) Quad-core, (c) Hexa-core, (d) Octo-core on multi-core cluster variation architectures	42

HIGH PERFORMANCE PARALLEL ASSOCIATION RULE MINING ON MULTI-CORE CLUSTER

INTRODUCTION

In 2011, about 1.8 zettabytes (1.8 trillion gigabytes) of data will be created. By 2020 the world will generate 50 times the amount of information (Gantz and Reinse, 2011). The amount of data continues to grow at a startling rate. From social networking, mobile device applications, website server logs, RFID sensor event data, point-of-sale transactions and credit card purchases. However, raw data by itself does not provide much information. The challenge is to convert the huge of raw data into valuable information - in the acceptable time - using data mining applications.

Association rule mining, a popular data mining technique, has been deployed in many decision support systems for years. It extracts significant patterns and then generates some interesting rules from large database. Association rule mining and big data make it possible for human societies to right decisions and bring enormous benefits to the social. This is substantially important in many domains such as bioinformatics, analysis of medical, scientific, and commercial data. 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, association rule mining applications can get benefits from the use of parallel computing systems to improve performance. Although parallel association rule mining algorithms handles large datasets but faces challenges like data decomposition and I/O disk minimization, work load balancing and communication or synchronization minimization that needs attention to achieve high performance.

This thesis proposes strategies to improve performance of a parallel association rule mining application on large cluster. In order to achieve a good performance, a good scheduling of parallel applications that usually consist of computation, communication, and possibly I/O

activities, is very important. This thesis addresses the three issues by presenting scheduling schemes. It consists of three strategies to support and speed up parallel association rule mining. First, improving I/O performance by selection proper data distribution and scheduling of parallel I/O data transfers on system. Second, provide the scheduling strategy to efficiently perform the execution task of association rule mining. Finally, managing task of communication by using the proper scheduling of message communication that consider the right sequence of message transmission, message size, and interconnection network utilization. Then, the proper scheduling strategies of the I/O, communication, and execution of the subtasks of parallel data mining will be linked together to improve the performance called Unified I/O, Communication and Execution Scheduling (U-ICE).

OBJECTIVES

The objective of this work is to improve performance of parallel association rule mining on a computing cluster environment.

1. To improve I/O activities performance using I/O scheduling

2. To improving execution task performance using computation scheduling

3. To improving communication task performance using communication scheduling

Scope of This Work

1. This work proposes the strategy to perform the task of association rule mining that consist of I/O tasks, execution tasks and message communication tasks.

2. The parallel FI-growth algorithm will be used to demonstrate the proposed concept.

3. We use a tool to evaluate the merit of each strategy on a real task graph obtained from the application on a target multi-core cluster system.

Benefits

1. Provide various scheduling strategies to efficiently perform the task of large-scale association rule mining.

2. Provide a tool to evaluate the merit of each strategy on a real task graph obtained from the application on a target multi-core cluster system.

LITERATURE REVIEW

This chapter presents a preliminary of basic concepts about association rule mining, the details of the FI-growth algorithm, and its design and parallel implementation. The parallel FI-growth algorithm will be used to demonstrate the proposed concept and computing cluster environment that is a target system described subsequently. Also, this chapter presents a literature survey of various sequential association rule mining algorithms and the exploiting parallelism to the association rule mining on different hardware platforms. We also present techniques are used for enhancing performance of parallel association rule mining.

Background

1. Basic Association Rule Mining Concept

Association rule mining, one of the most important techniques of data mining, is the process of extracting the novel significant or interesting correlations, frequent patterns, associations among set of items in the large volumes of transaction databases. The concept of association rules was first introduced by Agrawal *et al.* (1993) for market basket analysis. This process analyses costomer buying behaviors by discovering correlation between products that customers place in their shopping baskets. In brief, an association ruls is an implication of the form $X \Longrightarrow Y$, where X and Y are two itemsets, and $X \cap Y = \varnothing$. Let a transaction database D, where each transaction $T \in D$ is a set of items, $X \Longrightarrow Y$ expresses that whenever a transaction T contains X then T probably contains Y also. There are two important basic measures for association rules, support and confidence. The rule's support is the joint probability of a transaction containing both X and Y at the same time and is defined as $\sigma(X \cup Y)$. The rule confidence is the conditional probability that a transaction contains Y given that is contains X and defined as $\sigma(X \cup Y) / \sigma(X)$. Usually thresholds of support and confidence are predefined by users to prune those rules that are not so interesting or useful. The two thresholds are called minimum support and minimum confidence respectively. A rule or itemset is frequent if its support is greather than or equal to a

minimum support and a rule is strong if the confidence is greather than or equal to a minimum confidence.

2. FI-Growth Algorithm

Frequent pattern mining has been a focused theme in data mining research for over a decade. The FP-growth algorithm (Han *et al.*, 2000) is a new generation of frequent pattern mining that uses a compressed FP-tree structure for mining a complete set of frequent itemsets without candidate itemset generation. The algorithm is divided into two phases. (1) Construct an FP-tree that encodes the data set by reading the database and mapping each transaction onto a path in the FP-tree, while simultaneously counting the support of each item. (2) Extract frequent itemsets directly from the FP-tree using a bottom-up strategy to find all possible frequent itemsets ending with a particular item; equivalent to the suffix-based approach. This mining phase can run significantly faster if the FP-tree is small enough to fit into main memory. In practice, the more prefix paths are shared, the greater the compression of the FP-tree is. The size of the FP-tree is typically smaller if all items are ordered from highest to lowest support values. However, for very large databases, a lot of time is required to first sort the supports of 1-itemsets.

Similar to the FP-growth algorithm (Han *et al.*, 2000), the FI-growth algorithm (Amphawan and Surarerks, 2005) represents the data set as a prefix sharing tree, called an "FI-tree". The FI-growth algorithm does not only avoid time consuming operation of sorting the 1-itemsets as in FP-growth, but also provides an efficient operation for FI-tree manipulation. It commonly consists of two phases: FI-tree construction and mining. In the following subsections, an FI-tree representation and how it is constructed are briefly reviewed. Then the three steps of mining frequent itemsets with an FI-tree are presented; additional details and pseudo-codes are also provided in Amphawan and Surarerks (2005).

2.1. FI-tree representation

An FI-tree is a memory representation of the data set encoding, where each transaction is mapped onto the FI-tree with prefix sharing paths. In practice, the FI-tree structure consists of two structures in memory: a “header table” that has linkages to each item in an “items-tree”. Constructing an FI-tree requires scanning the database only twice: the first scan creates the header table, and the second scan creates the items-tree.

In order to show how an FI-tree is constructed, we use an example transaction dataset shown in Table 1. The first column in the table is the ordered transaction IDs, while the second one shows the list of items corresponding to the transaction. Note that the items in all lists must be in the same relative order.

Table 1 Example of a transaction database

TID	List of items
1	*a, c, d, e, f*
2	*a, b, e*
3	*c, e, f*
4	*a, c, d, f*
5	*c, e, f*

FI-tree construction first creates a header table having size corresponding to the number of unique items. During the first scan, each transaction is read to update the support values. Then, we prune items with supports that are less than a pre-defined minimum support, which is taken to be 2 (40%) in our example. During the second scan, FI-tree construction only considers the items that appear in the pruned header table (labeled below as *item:count*). Each transaction is read and additional paths in the FI-tree are created. The same sequences of items are grouped into the same sup-paths. The FI-tree constructed from the example transaction database is depicted in Figure 1.

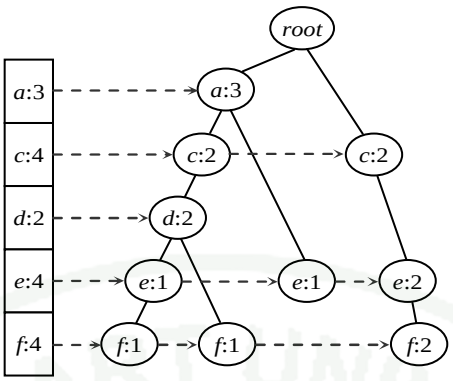

Figure 1 The initial FI-Tree structure: the header table (left) and the items-tree (right)

2.2. Frequent itemsets generation with FI-Tree

An advantage of the FI-tree is that we can directly generate and discover frequent itemsets without any further scanning of the database. This mining phase is composed of three steps. The first one is a branching step: the FI-tree is branched using an item frequency combination. The second step is subset finding: the algorithm attempts to explore all possible remaining paths in the FI-tree. The last is a pruning step used to eliminate uninteresting results.

2.2.1. Combining operation

A combining operation is a useful operation to merge two sub-trees, that is, the same sub-paths are grouped and their counts summed. For a new created sub-tree, it will either be added as a new child of root, or replace the old one if the root already has that child item. Let *tree*(*a*) be a tree (or sub-tree) that has a as a root node, and © refer to a combining operation. The combining operation has the following properties.

i) *Self-reflective property*: *tree*(*a*) © *tree*(*a*) is equal to *tree*(*a*) itself.

ii) *Commutative property*: *tree*(*a1*) © *tree*(*a2*) is equal to *tree*(*a2*) © *tree*(*a1*).

iii) *Associative property*: (*tree*(*a1*) © *tree*(*a2*)) © *tree*(*a3*) is equal to *tree*(*a1*) © (*tree*(*a2*) © *tree*(*a3*)).

For example, consider the item "*c*" of the header table in Figure 1. It has linkages to two sub-trees. The combining operation applied on these two sub-trees is depicted in Figure 2. The result (grey nodes) replaces the old one that is linked from *root*.

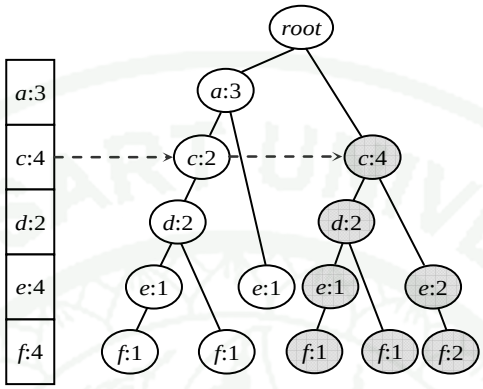

Figure 2 Result of combining operation applied to two sub-trees tree(c)

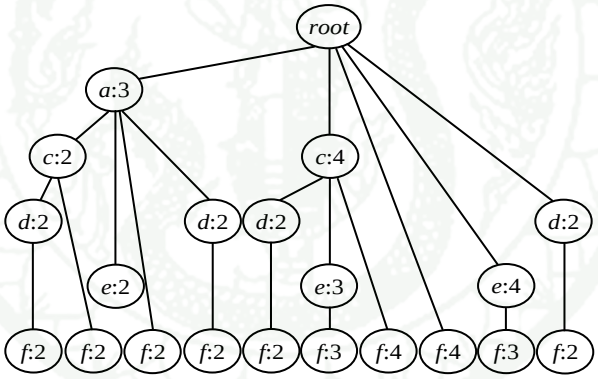

Figure 3 The final items-tree derived from the transaction database in Table 1

2.2.2. Mining process

As mentioned before, there are three steps in this phase.

Branching step: The first step is to sum the count of each item. From the initial FI-tree in Figure 1, we iteratively apply the combining operation to all sub-trees that are linked from the same item contained in the header table.

Subset finding step: The main goal of this step is to find all possible remaining paths. Consider each sub-tree of the root node; we first create linkages from the header table to all nodes, and then recursively apply the combining operation to that sub-tree. After that, we discard all linkages, and iterate the process for finding subsets of the remaining sub-trees.

Pruning step: The last step is to remove nodes (and also paths) for itemsets that fall below the minimum support. For the example FI-tree in Figure 1, the final items-tree is depicted in Figure 3. We have pruned the items using minimum support equal to 40%. In this tree we can easily spot the frequent itemsets, such as $\{a : 3\}$, $\{ a, c : 2 \}$, $\{ a, c, d : 2 \}$, $\{a, c, d, f : 2\}$.

3. Parallel FI-Growth

Parallel FI-growth (Manaskasemsak *et al.*, 2007) used for demonstrate the proposed concept 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 multi-core cluster. The work flow of parallel FI-growth is shown in Figure 4.

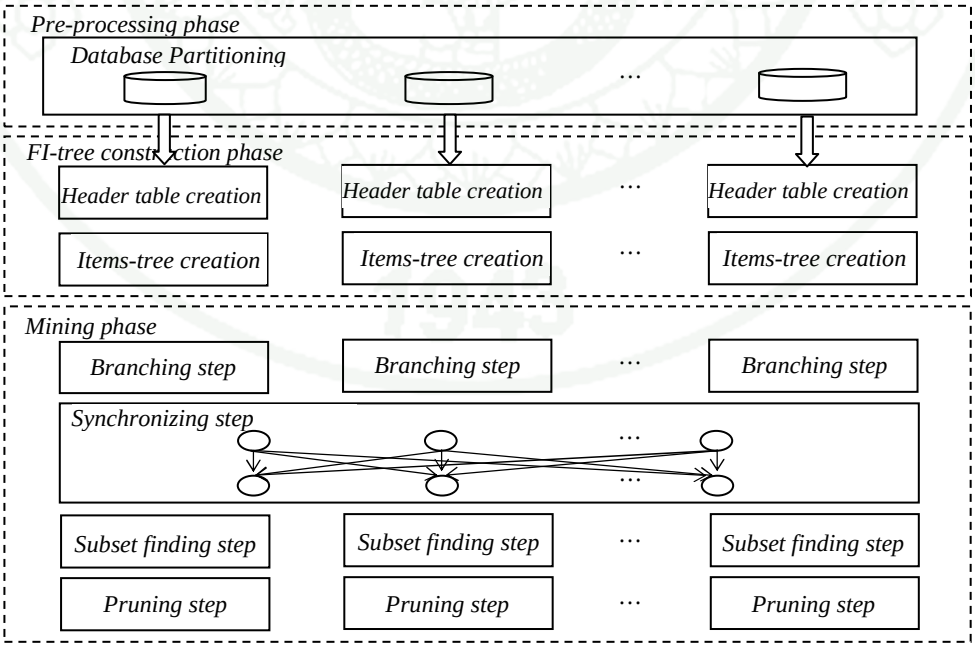

Figure 4 The parallel FI-growth work flow

The parallel FI-growth work flow consists of 3 phases, which are preprocessing phase, FI-tree construction phase, and mining phase. We first partition the transaction database into several portions, and distribute them to different processors for computation. 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. Hierarchical minimum support

In the pre-processing phase, transaction database is partitioned and portions are distributed to compute nodes. The partitioning has an effect on the local counts, namely, a decrease of support values of items residing in each partition. Consequently, some items may be pruned out if local pruning is performed, although their global support counts would exceed the required *min_sup* value. The more partitions created, the more items may be pruned out locally. This pruning directly affects the correctness of final results. However, there are two solutions to avoid such a problem.

3.1.1 After creating each local header table, all processors synchronize their lists of item counts, and then merge those lists to obtain global support values. Only the items whose global support is less than the *min_sup* are pruned out.

3.1.2 Incorporate a hierarchical minimum support technique, utilizing two values of minimum support: min_sup_{L1} is defined and used to prune the local header table while min_sup_{L2} is defined to prune the local items-tree. Consequently, min_sup_{L1} is set to be less than min_sup_{L2}.

In practice, the first approach provides the most correct result; however, it requires a costly all-to-all broadcast process. The second one gives an approximate pruning that avoids the significant communication overhead of the first one, at the expense of some errors depending on

the value of min_sup_{L1} and dataset partitioning. In the implementation, we use the second approach.

3.2. Parallelization

In the pre-processing phase, transaction database is equally partitioned and partitions are assigned to all processors. For example, to process the example database in Figure 1 using two processors, we could assign TIDs 1, 2, 3 to the first processor p_0, and assign TIDs 4, 5 to the second processor p_1.

In the distributed FI-tree construction phase, suppose we assign min_sup_{L1} a value of 1 (or 20%) and min_sup_{L2} a value of 2 (or 40%). The resulting initial local FI-trees are depicted in Figure 5.

In the mining phase, the combining operations during the branching step can be independently performed by each processor. Figure 6 depicts the evolution of the FI-trees after the branching step. The synchronization of sub-trees' exchange occurs between the processors. Then, the subset finding and pruning steps can be independently run on each processor.

3.3. FI-Tree synchronization

The synchronizing step begins with all processors exchanging their local header tables. The combined header table is used to determine which local sub-trees should be sent to which target processors.

3.3.1 *Exchanging of local header table*: This process is used to create a global header table on each processor. To reduce the communication over-head, only the list of items is broadcast to other processors. The left-most column in Figure 7 shows the resulting global header tables of p_0 and p_1.

3.3.2 *Sending of local sub-tree*: After each processor has its own global header table, it will determine which local sub-tree(s) should be kept, and which should be sent to the target processors and deleted afterward. Let *tree*(x) be a sub-tree having item x as root, and |*tree*(x)| be the number of nodes within that sub-tree. Consider any two sub-trees *tree*(x) and *tree*(y); if item x comes before item y in the sequence of items, then the former will have more possible paths to expand than the latter so that |*tree*(x)| is in general greater than |*tree*(y)|. For the example shown in Figure 6, the sequence of items is < a, b, c, d, e, f >; then |*tree*(a)| is probably greater than |*tree*(b)|, or |*tree*(c)| is probably greater than |*tree*(d)|, etc.

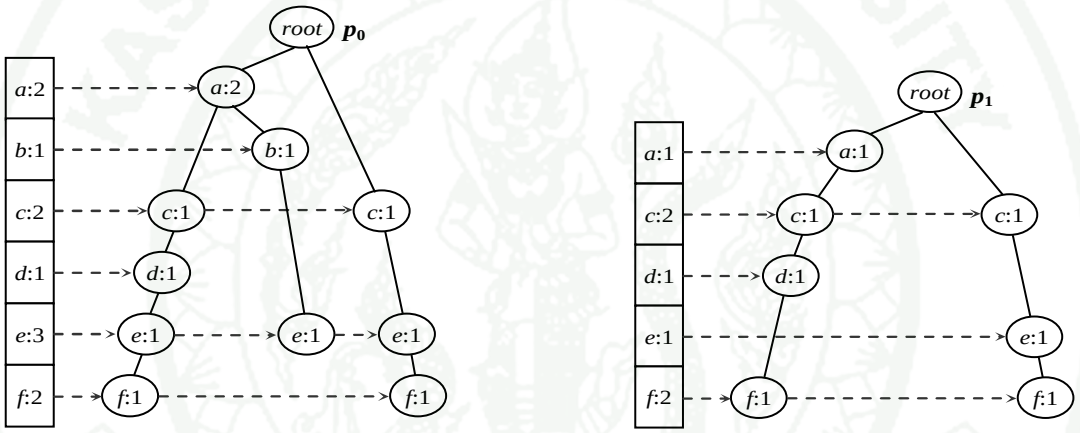

Figure 5 The initial FI-Tree structures residing at the processor p_0 (left) and processor p_1 (right)

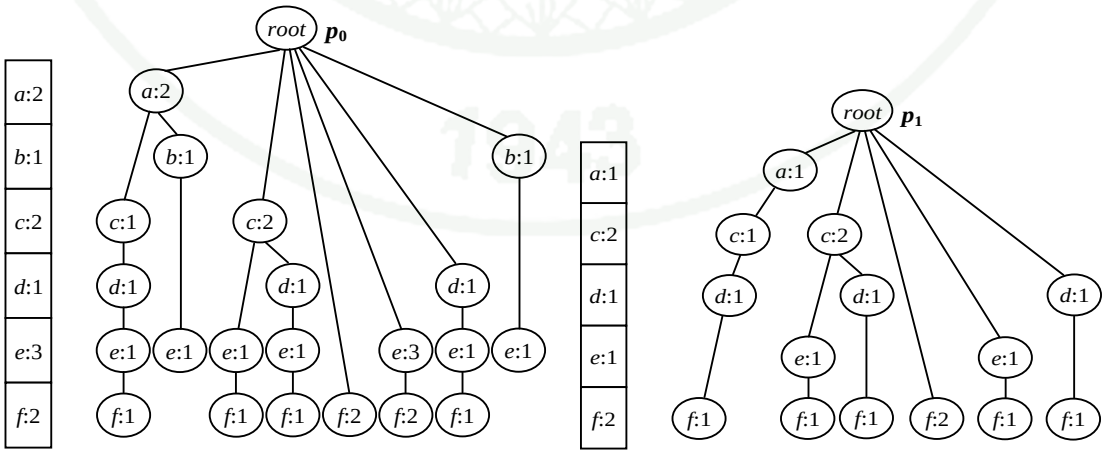

Figure 6 The evolution of FI-Tree structures after the branching step

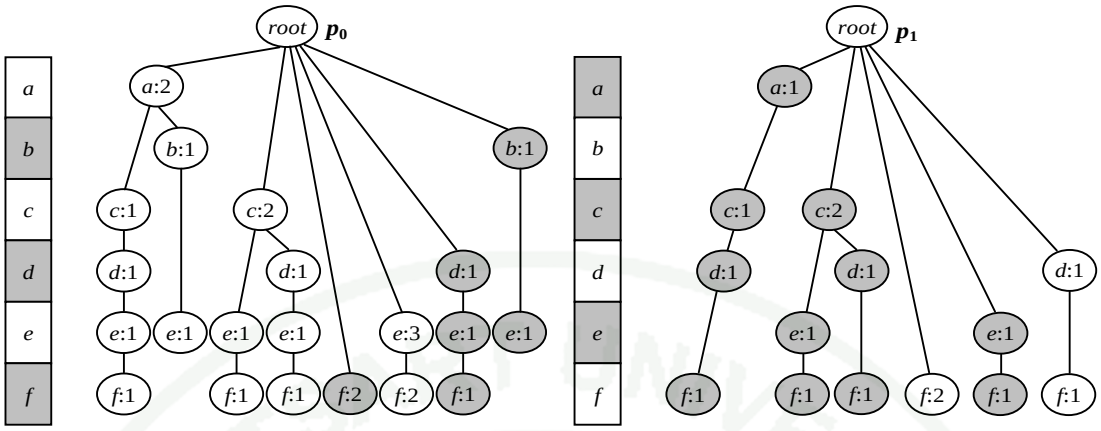

Figure 7 The evolution of FI-Tree structures during the synchronizing step. Here only grey color sub-tree will be moved to the other processor.

Since we attempt to balance the computing load between processors, we use a simple round robin heuristic to choose which sub-trees should be mined at which processor. We take the remainder of the item's order in the sequence of items modulo the number of processors to assign a sub-tree to a corresponding processor. For the example in Figure 7, since we begin counting the sequence of item *a* from 0, sub-tree *tree*(*a*), *tree*(*b*), *tree*(*c*), *tree*(*d*), *tree*(*e*), and *tree*(*f*) will be assigned to processor p_0, p_1, p_0, p_1, p_0, and p_1, respectively.

4. Multi-Core Cluster

Multi-core means to integrate two or more processing cores on the same chip (Lei *et al.*, 2007). Multi-core processor is also known as Chip Multiprocessor (CMP). Since multiple processor cores are integrated onto a single integrated circuit die or onto multiple dies in a single chip package. The multiple cores can run multiple instructions at the same time so increasing overall speed for programs responsive to parallel computing.

Two typical multi-core system designs are illustrated in Figure 8. First, a NUMA based dual-core system shown in left box, each core has its own L2 cache and all the cores on the same chip share the memory controller and local memory. Cores can also access remote memory, although local memory access is much faster. Second, a bus based dual-core system that used in

this thesis, shown in the right box. All the cores on the same chip share the same L2 cache and memory controller, and all the cores access the main memory through a shared bus.

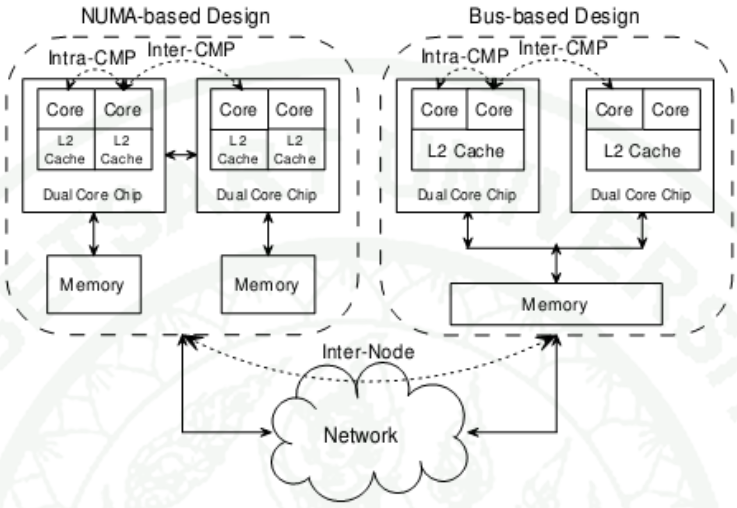

Figure 8 Illustration of typical multi-core clusters

Source: Lei *et al.* (2007)

Multi-core processors have been deployed in cluster system. The multi-core cluster system is connected together, using a hierarchy of communication networks. There are three levels as shown in Figure 8. First, an intra-CMP is the communication between two cores on the same chip is referred to as *inter-core communication* in this thesis. Second, an inter-CMP is the communication across chips but within a machine is referred to as *inter-processor communication* in this thesis. Finally, an inter-node is the communication between two cores on different machines is referred to as *an inter-machine communication*.

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.

5. Parallel FI-Growth Application Runtime Characteristic

In order to improve performance of a parallel application, it is essential to characterize the runtime behavior of this application. This understanding of a parallel FI-growth application runtime can lead to the design strategy for minimizing overall runtime. We investigate how the results vary when we change the number of processors used in the execution. Our measures of interest include the overall program run time which composes of the I/O time, the computing time and the synchronization or message communication time. We mined a 16 million transaction database by using the parallel FI-growth program (Manaskasemsak *et al.*, 2007). We utilized the standard "IBM synthetic data generator" (Srikant, 2006) to synthesize a transaction database. We used 1000 unique items to create 16 million records; each has average transaction length of ten. The data file size is 582 Mbytes. The result is shown in Figure 9.

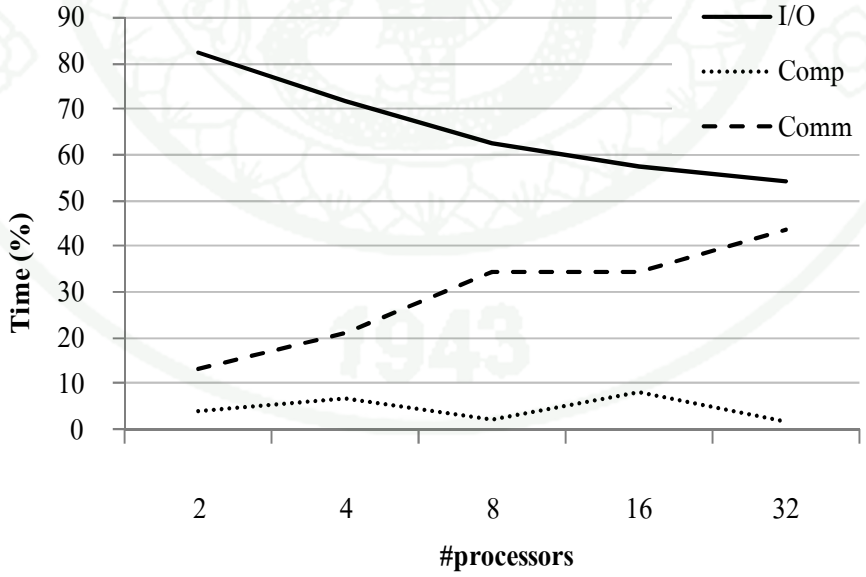

Figure 9 The trends in term of percentage of the I/O time (I/O), computing time (Comp), and communication time (Comm)

Figure 9 illustrates the percentage of run times spent in three parts for parallel FI-growth program using 2, 4, 8, 16, and 32 processors respectively. The result shows that the I/O time and communication time takes up a great percentage of total run time while the computation time takes up a small percentage. When we increase amount of processor, the percentage of I/O time is decreased because the I/O load can be spread to the added coordination of processors whereas message communications between the coordination of processors is increased lead to increase the percentage of communication time. From this result, we found that challenges like data distribution and I/O disk minimization, work load balancing and communication minimization that need attention to achieve high performance. The possible strategies to optimize the performance of parallel association rule mining algorithm consist of improving I/O using I/O scheduling, improving computation using computation scheduling, and improving communication using communication scheduling.

Related Works

This section presents a comprehensive literature survey of various sequential association rule mining algorithms and the exploiting parallelism to the association rule mining on different hardware platforms. We also present these techniques are used for enhancing performance of parallel association rule mining algorithms.

1. Association Rule Mining Algorithm

The *Apriori* algorithm (Agrawal *et al.*, 1993; Agrawal and Srikant, 1994) has emerged as the best classical algorithm for association rule mining since 1994. This algorithm is based on the generation of candidate itemsets of length k from itemsets of length k-1 from the previous pass. Then it prunes the candidates if they are infrequent enough, i.e., their frequency or support is less than a user-specified minimum support value. We call these processes the “generate-and-test” paradigm. Apriori algorithm was proposed to first find the smallest frequent itemsets (1-itemsets) and then expand to larger itemsets, while the *A-Closed algorithms* (Pasquier *et al.*, 1999)

performed in the opposite way. The disadvantages of these algorithms are: (1) candidate itemsets generation requires a very long computation time, (2) the database must be scanned several times so the I/O requirement is rather intensive. Park *et al.*, (1995a) proposed the *Direct Hashing and Pruning* (*DHP*) algorithm 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. The *Partition* algorithm is proposed by Savasere *et al.*, 1995 which reads the database at most two times to generate all significant association rules. In the first scan of the database, it divides the database into a number of non overlapping partitions, where each partition is read and the global candidates are generated by merging all local frequent itemsets. The set of global candidates is a superset of all frequent itemsets so it may contain itemsets that are not frequent. During the second scan, the actual supports for these itemsets are generated and frequent itemsets are indentified.

Han *et al.* (2000) presented a new algorithm of mining association rules called the *FP-growth (Frequent Pattern growth)* algorithm. This algorithm finds a complete set of frequent itemsets by avoiding candidate itemsets generation by using a Frequent Pattern tree (FP-tree) structure. FP-tree is a tree structure that contains all prefixes of items in transactions. The basic idea of the pattern growth approach is to grow a pattern from its prefix. Similar to the FP-growth algorithm (Han *et al.*, 2000), *FI-growth* (*Frequent Item growth*) algorithm (Amphawan and Surarerks, 2005) represents the data set as a prefix sharing tree, called an "FI-tree". The FI-growth algorithm does not avoid time consuming operation of sorting the 1-itemsets as in FP-growth and they have shown that the complete set of frequent itemsets can be generated by using a single tree. The FP-growth and FI-growth algorithms were proposed to reduce cost of candidate itemsets generation and several times database scanned. These algorithms need at most two database scans; but need to fit all data structures in main memory. These algorithms are classified as shown in the Table 2. Further improvements of mining methods have been proposed *Tree projection* (Agarwal *et al.*, 2001), *Closet+* (Wang *et al.*, 2003), *Carpenter* (Pan *et al.*, 2003), *CHARM* (Zaki and Hsiao, 2002, 2005), *IFP-growth* (Lin *et al.*, 2011).

Table 2 Classification of association rule mining algorithms

Algorithm	Data layout	Search technique	Data structure	Read input	Parallel algorithm
Apriori	Horizontal	Bottom-up	Hash tree	Several times	CC, CD, CCPD,PCCD
Closed	Horizontal	Top-down	Hash tree	Several times	TFP
DHP	Horizontal	Bottom-up	Hash tree	Several times	PDM
Partition	Vertical	Bottom-up	None	Two times	PPAR
FP-growth	Horizontal	Bottom-up	FP-tree	Two times	MLFPT, PFP-growth
FI-growth	Horizontal	Bottom-up	FI-tree	Two times	Parallel FI-growth

Many parallel and distributed algorithms proposed for speeding up the computation time of association rule mining algorithms are either generation-and-test approach or pattern growth approach. In generation-and-test approach, Agrawal and Shafer (1996) proposed three parallel approaches for mining association rules on a distributed-memory machine: *Count Distribution (CD)*, *Data Distribution (DD)*, and *Candidate Distribution (Cand Dist)*. *CD* was shown to have superior performance among these three algorithms. Zaki *et al.* (1996) proposed two algorithms *Common Candidate Partitioned Database (CCPD)* and *Partitioned Candidate Common Database (PCCD)* to improve the further speed and efficiency of the approaches proposed in Agrawal and Shafer, (1996). Han *et al.* (1997) proposed *Intelligent Data Distribution (IDD)* algorithm, *Hybrid Distribution (HD)* algorithm which parallel and distributed methods based on Apriori. *Parallel Data Mining (PDM)* algorithm (Park *et al.*, 1995b) is a parallel version of the DHP (Park *et al.*, 1995a) 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. *TFP* (Jianyong *et al.*, 2005) is parallel algoritm of the *Closed* algorithm for mining top-k frequent itemsets. In a pattern growth approach, most of these algorithms exploit the parallelism by partitioning the database into several portions and then build local frequent pattern tree of each portion in parallel, *Multiple Local Frequent Pattern Tree (MLFPT)* algorithm (Zaiane *et al.*, 2001), *Parallel FP-tree (PFP-tree)* (Javed and Khokhar, 2004), *Load Balancing FP-tree (LFP-tree)* (Yu *et al.*, 2007), *parallel FI-growth* (Manaskasemsak *et al.*, 2007), *Parallel Pattern tree*

(PP-tree) (Tanbeer *et al.*, 2009). Some algorithms build a global frequent pattern tree for entire database by parallelizing the process of tree construction, such as *Tree Partition based FP-tree* (Dehao *et al,* 2006) and Buehrer *et al.* (2007).

Hardware platform has two dominant architectures: *Shared Memory (SMP) Architecture*, *Distributed Memory (DMM) architecture*. In shared memory architecture, all processors access common memory and communicate through shared memory. In distributed memory architecture each process has its own local memory and communicates through message passing. Shared memory system is addressed in the parallel and distributed implementation of association rule mining such as *MLFPT* (Zaiane *et al.*, 2001), *CCPD* and *PCCD* (Zaki *et al.*, 1996), *Tree partition based FP-tree* (Dehao *et al.*, 2006). Whereas distributed memory system is addressed in the parallel and distributed implementation of association rule mining such as in *PDM* (Park *et al.*, 1995b), *CD*, *DD*, and *Cand.Dist.* (Agrawal and Shafer, 1996), *IDD* and *HD* (Han *et al.*, 1997), and *PFP-tree* (Javed and Khokhar, 2004).

There are two broad strategies for parallelizing association rule mining algorithms (Skillicorn, 1999). Firstly, task parallelism (or control parallelism) is exploited by having each process execute different operations on the database such as in *DD* and *Cand. Dist.*(Agrawal and Shafer, 1996), *PCCD* (Zaki *et al.*, 1996), and *HD* (Han *et al.*, 1997). Secondly, data parallelism can be employed by having a set of processes execute the same algorithm in parallel on difference partitions of the database such as in *CD* (Agrawal and Shafer, 1996), *PDM* (Park *et al.*, 1995b), *MLFPT* (Zaiane *et al.*, 2001), and *CCPD* (Zaki *et al.*, 1996).

The performance is the key issue while mining large dataset. Many methods also proposed to improve the efficiency. Efficiently partitioning database methods for optimal load-balance have been proposed in *LFP-tree* (Yu *et al.*, 2007), *Multi-core Apriori Transaction Identifiers (MATI) algorithm* (Kun-Ming, 2011). Minimum communication overheads have also been reported in *PFP-tree* (Javed and Khokhar, 2004) and *Tree Partition based FP-tree* (Dehao *et al,* 2006). *Simple placement (SPP)* strategy to improve execution time by using memory

placement policy had been proposed in Parthasarathy *et al.* (2001). *PP-tree* (Tanbeer *et al.*, 2009) proposed a new data structure which reduces the I/O cost by scanning the database only once. From above literature survey, the parallel association rule mining design to achieve good performance on parallel computing systems are classified on three main components: the hardware architecture, type of parallelism, and improving performance techniques shown in Figure 10.

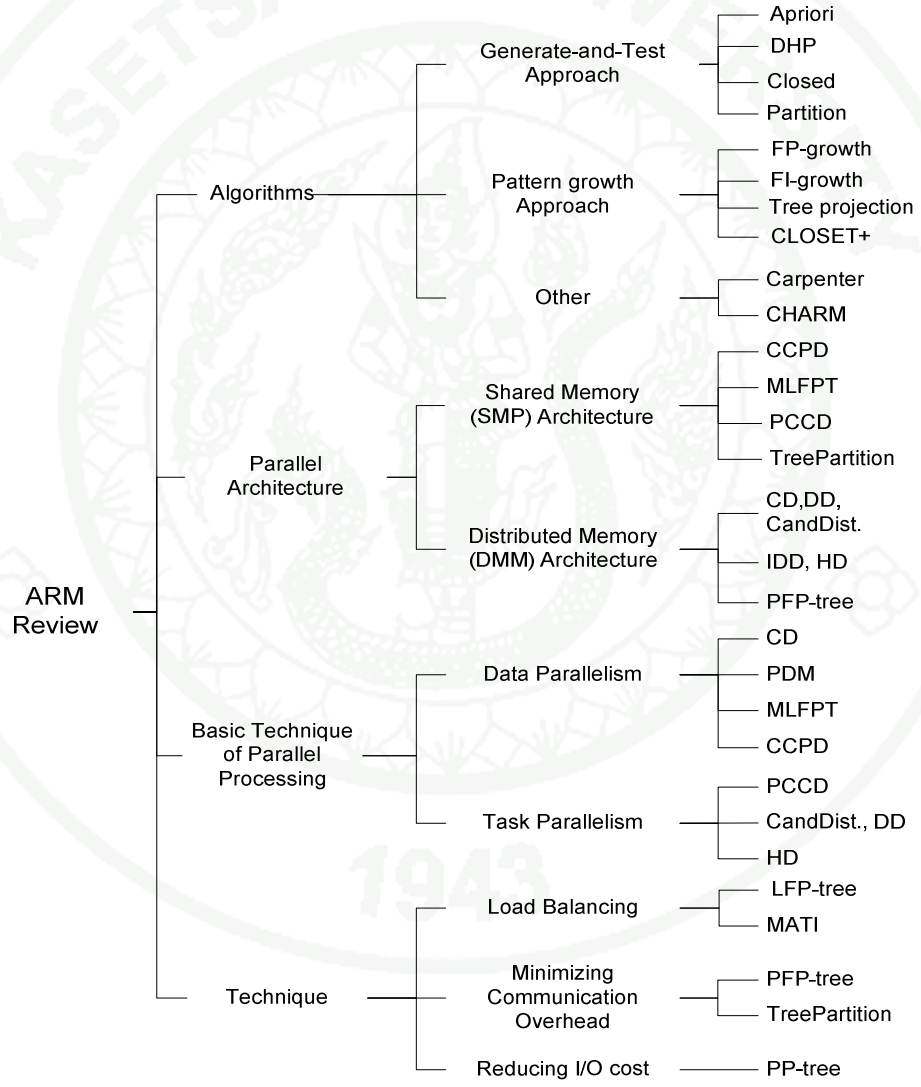

Figure 10 Association rule mining review

MATERIALS AND METHODS

Materials

1. A cluster of x86-64 based SMP machines named “Tera cluster”, depicted as Figure 12. Each machine in the cluster we used an Intel Xeon processor 5000, comprising two Dual-core CPUs. Each core has a clock frequency of 3.20GHz and each node has 4GB main memory, and two 80GB SATA disks. The machines in the cluster are connected by Gigabit switch. All machines are equipped with the Linux-based operating system.

2. A computer notebook with Core2 Duo 1.66 GHz CPU, memory 2GB , hard disk 140 GB, and Windows Vista operating system.

Methods

1. Models and Problem Statement

In this section, we introduce several models that will be used in this thesis, and then provide the problem formulation.

1.1. Application model

In this thesis, the parallel association rule mining can be represented by a *Directed acyclic graph (DAG)*, which is used as an input to the scheduling simulator.

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 be a set of nodes $\{ v_0, v_1, \ldots, v_n \}$ and each node represents one task. $E \subseteq V \times V$ be the edge set that defines communication and precedence constraints relations for all nodes in V. Each directed edges, $(v_i, v_j) \in E$ in the DAG connecting nodes v_i and v_j represents the communication and precedence constraints between tasks T_i and T_j .

Let the weight of an edge $W_E(v_i, v_j)$ denote the communication cost between two tasks. Each node of the task graph has a weight $W_V(v_i)$ that represents the computing cost. An example is given in Figure 11.

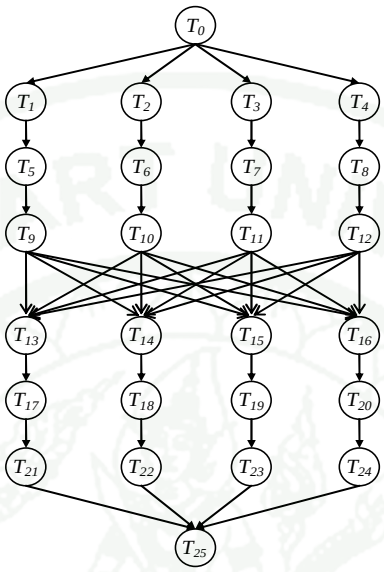

Figure 11 The parallel FI-growth task graph

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. Each node of the task graph has a weight that to be the computing cost that includes memory access time. For communication contention, a task cannot receive data from all its predecessors simultaneously, but it receives data sequentially from the same communication channel. The communication cost between two tasks assigned to the same processor is supposed to be zero.

1.2. System model

The parallel FI-growth application were run on a cluster of x86-64 based SMP machines named “Tera cluster”, depicted as Figure 12. Each machine in the cluster we used an Intel Xeon processor 5000, comprising two Dual-core CPUs. Each core has a clock frequency of 3.20GHz and each node has 4GB main memory, and two 80GB SATA disks. The machines in the

cluster are connected by Gigabit switch. All machines are equipped with the Linux-based operating system.

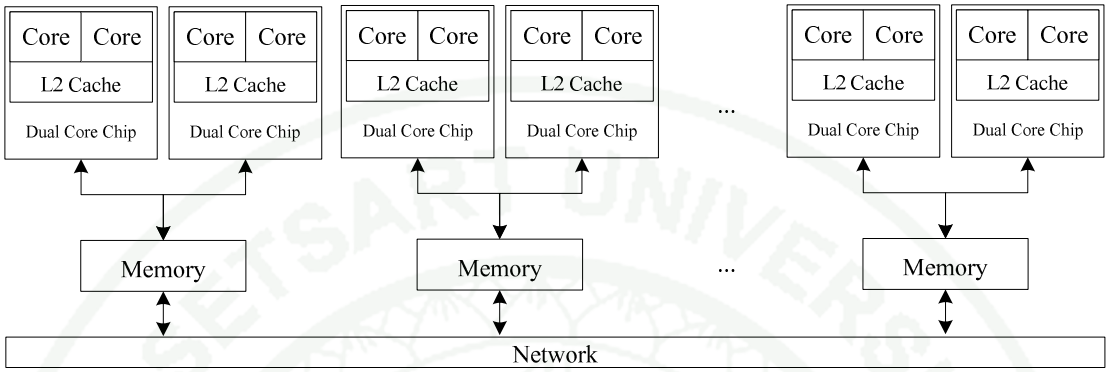

Figure 12 A multi-core cluster system

In this thesis, a multi-core cluster system that shown in Figure 12, is employed as the target architecture. A multi-core cluster system is a set machine $M = \{m_1, m_2, \ldots, m_i\}$ of $|M|$ machines, and each machine m_i includes a set of processor $P^i = \{p_1^i, p_2^i, \ldots, p_j^i\}$. Each processor p_j^i of machine m_i has a set of core $C^{ij} = \{c_1^{ij}, c_2^{ij}, \ldots, c_k^{ij}\}$. The total number of processors and the total number of cores are $|P| = \sum_{i=1}^{|M|} |P^i|$ and $|C| = \sum_{i=1}^{|M|} \sum_{j=1}^{|P|} |C^{ij}|$ respectively. The 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 thesis, we assume that the bandwidth between cores is 20 Gbps, bandwidth between processors is 10 Gbps, and bandwidth between machines is 1 Gbps. The multi-core cluster system can be represented by 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 shown in Figure 13.

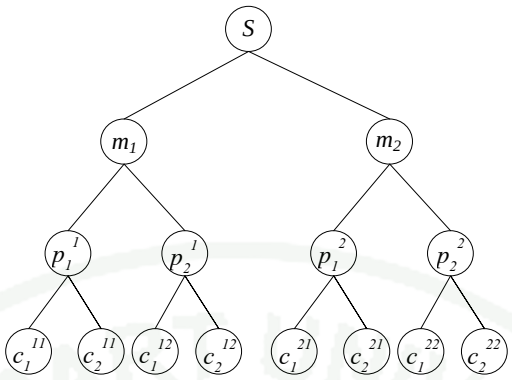

Figure 13 System model

1.3. Database partitioning

Association rule mining finds interesting association or correlation relationship among a large set of data items in the large volumes of transaction databases. Let a transaction database D be a set of n transactions such that $D = \{T_1, T_2,..,T_n\}$, where $T_i = \text{I}$ and $\text{I} = \{i_1, i_2,..,i_m\}$ be a set of m attributes called items. Each transaction in D has a unique transaction ID (TID) and contains a subset of the items in I. An example transaction dataset is given in Table 1. In this thesis, the transaction database is apportioned amongst the processes, by horizontally segment the dataset into sets of transactions which are non-overlapping and equal size. Let $\text{B} = \{B_1, B_2, ..., B_l\}$ be a set of l partitioned dataset called "data-block". The data-block should also be assigned to processing nodes.

1.4. Problem statement

Based on models and concepts in previous mention, we further clarifying the problem as follows: Given a DAG $G = (V, E)$ of a parallel data mining application, a multi-core cluster system which connected together, using a hierarchy of communication networks, with $|M|$ machines, $|P|$ processors and $|C|$ cores, and a transaction database D with $|B|$ data-blocks. We study the following three problems:

1.4.1. How to improve I/O operations performance of parallel FI-growth application?

1.4.2. How to design the scheduling strategy for parallel FI-growth application?

1.4.3. How to utilize interconnection network that will yield a great performance enhancement to parallel FI-growth application?

2. Proposed Strategies

In order to further enhance the performance of a parallel FI-growth data mining application, this thesis proposes several strategies, consisting of data distributing and I/O operations scheduling strategy, scheduling strategy for executing task of parallel FI-growth application, scheduling strategy of message communications and then unify the I/O, computation, and communication scheduling together to achieved a much better performance. After formulating these strategies, the next step is to test which strategies will perform the best for our target system which is the large multi-core cluster.

2.1. Improving I/O: distributing data and scheduling I/O operations

Many association rules mining applications are strong I/O intensive, so I/O task has become a bottleneck in the computing systems due to its poor performance. In this thesis, the issues concerning I/O improving is not only the size of the data to be mined, but also data distribution. We evaluate three simple data distribution strategies consist of round robin, range, and random with different number of data-blocks for parallel FI-growth algorithm. A multi-core cluster system and data preparation which mention is previous chapter, are used in the simulator.

We first partition the transaction database into several data-blocks, which are non-overlapping. The data-blocks obtained from partitioning are distributed and stored in different I/O devices. For distribution of data-blocks, we use three fundamental types, round robin, range, and

random which are an obvious type. Figure 14 provides a simple illustration of three data distributions (for four processors).

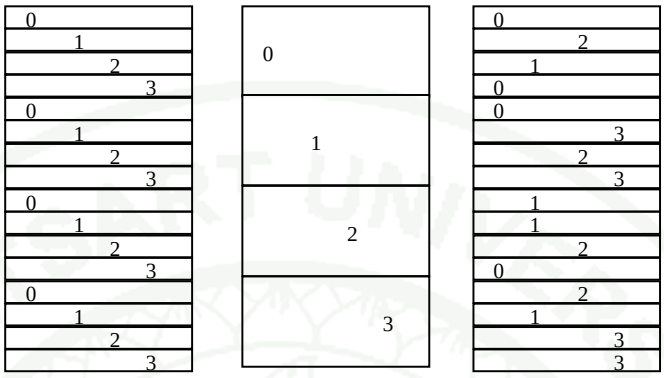

Figure 14 Three data distributions

For computing processors, data-blocks to be mine are assigned to different computing processors by using the same three simple data distribution strategies. By combining these variations of data distributions of server and client, we obtain nine basic data distribution strategies as shown in Table 3.

Table 3 Possible data distribution strategies

Data distribution		Server		
		RoundRobin	Range	Random
Client	RoundRobin	I	II	III
	Range	IV	V	VI
	Random	VII	VIII	IX

From distributed data, the scheduling problem can be formulated by a matrix. Given a set of computing processor $Cp = \{Cp_1, Cp_2, ..., Cp_n\}$, a set of I/O device $Io = \{Io_1, Io_2, ..., Io_m\}$ and a set of data-block $B = \{B_1, B_2, ..., B_l\}$. Let CpB be a matrix $n \times l$ containing distributed-data value by a variable CpB_{ik} equals to one when the data-block B_k is computed by processor Cp_i, and

IoB be a matrix $m \times l$ containing distributed-data value by a variable IoB_{jk} equals to one when the data-block B_k that is stored on I/O device Io_j. These matrices are used as an input to the scheduling simulator. It is assumed that each data transfer requires a specified pair of one processor and the other I/O device from two given sets of processors and I/O devices. Each processor and each I/O device may perform at most one transfer at any given time and each processor can communicate via a direct link with each I/O device. The output of I/O scheduling algorithm is a list of processor-I/O device pair that is scheduled and the completion time of data transfer requests.

During our I/O scheduling, we take both the available time of processors or I/O devices and the completion time of data transfer requests. The available time of processor or I/O device is the time that processor or I/O device can execute a new data transfer request. Each processor Cp_i attend to an available time $CpTime_i$ and each I/O device Io_j attend to an available time $IoTime_j$. The I/O scheduling algorithm proceeds as Figure 15.

I/O scheduling algorithm:

array of list CS_r ; // an array of list of data transfer requests of each list r

list $CS_r \leftarrow$ mapping (*CpB*, *IoB*);

repeat until all *CS* become empty

for $i = 1..n$

gets the first data transfer request (Cp_i, Io_j, B_k) of CS_r

the new pair (Cp_i , Io_j) will be scheduled at time t,

where $t = \max(CpTime_i, IoTime_j)$

update $CpTime_i$: $CpTime_i = t$ + (dataBlock size/IO bandwidth)

update $IoTime_j$: $IoTime_j = t$ + (dataBlock size/IO bandwidth)

remove first data transfer request (Cp_i, Io_j, B_k) from CS_r

end for

end repeat

Figure 15 The pseudo code of I/O scheduling algorithm

2.2. Execution scheduling of task on processors

In order to obtain a good speedup, the right execution scheduling should be employed. The parallel FI-growth data mining is analyzed and used to demonstrate the proposed concept. A directed acyclic graph $G = (V, E)$ of a parallel FI-growth data mining application, a multi-core cluster system which mention is previous chapter, are used as an input to the scheduling simulator. It is assumed that tasks are static and known beforehand. In order to design the scheduling strategy for parallel FI-growth application, the execution is divided into two steps described in the next subsection.

2.2.1. Strategies for ordering the task

A sequence of task for scheduling is created by assigning them priority. In this thesis, we use three ways to determine the priorities of nodes.

Smallest-numbered available task first or Left-to-Right (L-R) : Under this priority, a scheduling list from task graph in Figure 11 is T_0, T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14}, T_{15}, T_{16}, T_{17}, T_{18}, T_{19}, T_{20}, T_{21}, T_{22}, T_{23}, T_{24}, and T_{25}

Largest-numbered available vertex first or Right-to-Left (R-L) : Under this priority, a scheduling list from task graph in Figure 11 is T_0, T_4, T_3, T_2, T_1, T_8, T_7, T_6, T_5, T_{12}, T_{11}, T_{10}, T_9, T_{16}, T_{15}, T_{14}, T_{13}, T_{20}, T_{19}, T_{18}, T_{17}, T_{24}, T_{23}, T_{22}, T_{21}, and T_{25}

Top-to-bottom (T-B) : Under this priority, a scheduling list from task graph in Figure 11 is T_0, T_1, T_5, T_9, T_2, T_6, T_{10}, T_3, T_7, T_{11}, T_4, T_8, T_{12}, T_{13}, T_{17}, T_{21}, T_{14}, T_{18}, T_{22}, T_{15}, T_{19}, T_{23}, T_{16}, T_{20}, T_{24}, and T_{25}

2.2.2. Strategies for task mapping

We considered three criterions for mapping tasks to processing unit.

Earliest start-time : Tasks are allocated to a processing unit which allows the earliest start-time first.

Largest size of data transfer : Tasks are allocated by considering the size of data transfer from its parent nodes. The task is allocated to a same processing unit of its parent node which has largest size of data transfer.

Largest size of data transfer & earliest start-time : Tasks are allocated by considering the factors of both the start-time of processor unit and the size of data transfer from its parent nodes. At first we consider size of data transfer from its parent nodes and then follow by the earliest start-time.

By combining these variations of ordering and mapping strategies, we obtain nine basic scheduling strategies as shown in Table 4.

Table 4 Possible scheduling strategies

No.	Ordering	Mapping
1	L-R	Earliest start-time
2	L-R	Largest size of data transfer
3	L-R	Largest size of data transfer & earliest start-time
4	R-L	Earliest start-time
5	R-L	Largest size of data transfer
6	R-L	Largest size of data transfer & earliest start-time
7	T-B	Earliest start-time
8	T-B	Largest size of data transfer
9	T-B	Largest size of data transfer & earliest start-time

2.3. Optimization of communication overhead

Most parallel data mining applications usually generates a lot of communication messages. In this section, we propose that the proper scheduling of communication message that consider the right sequence of message transmission, message size, and interconnection network utilization will yield a great performance enhancement to parallel data mining application.

The parallel FI-growth data mining is analyzed and used to demonstrate the proposed concept. A directed acyclic graph $G = (V, E)$ of synchronizing step of a parallel FI-growth data mining application, a multi-core cluster system which mention is previous chapter, are used as an input to the scheduling simulator. It is assumed that the number of processors and the number of tasks are static and known beforehand. 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 16.

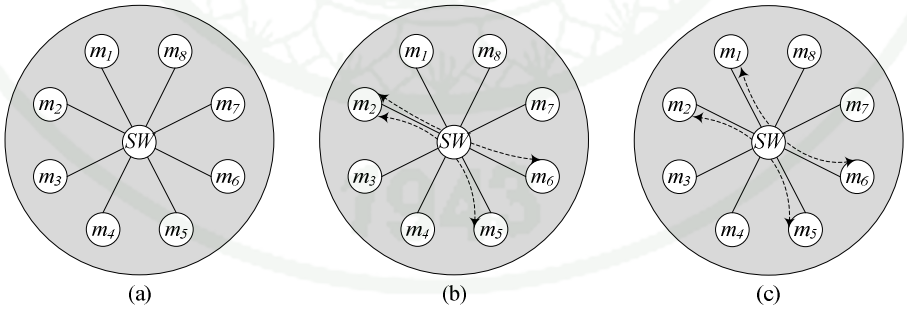

Figure 16 (a) Network model, (b) More than one communication sessions are initiated to a single target node. (c) The communication between a pair of node is contention free.

In Figure 16a, it is a fully connected network. Thus, every machine (*m*) has its own link with every other machine via switch (*SW*). Figure 16b, machine m_5 and machine m_6 need to send messages to machine m_2 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 16c. 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 order to further enhance the performance, we propose an approach called *Unified I/O, Communication and Execution Scheduling (U-ICE).* The concept is to combine I/O, computation, and communication scheduling together to achieved a much better performance. The U-ICE 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.

2.3.1. 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 $log_2 n$.

2.3.2. Communication step

In each partitioning step, all nodes in each partition are scheduled to send or receive message to/from all nodes within another partition. The number of communication steps of each partitioning step is equal to the number of nodes in the divided partition. The total number of communication steps is *n-1* for each node. Therefore, the total number of the exchange of data between all nodes in the system 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 17.

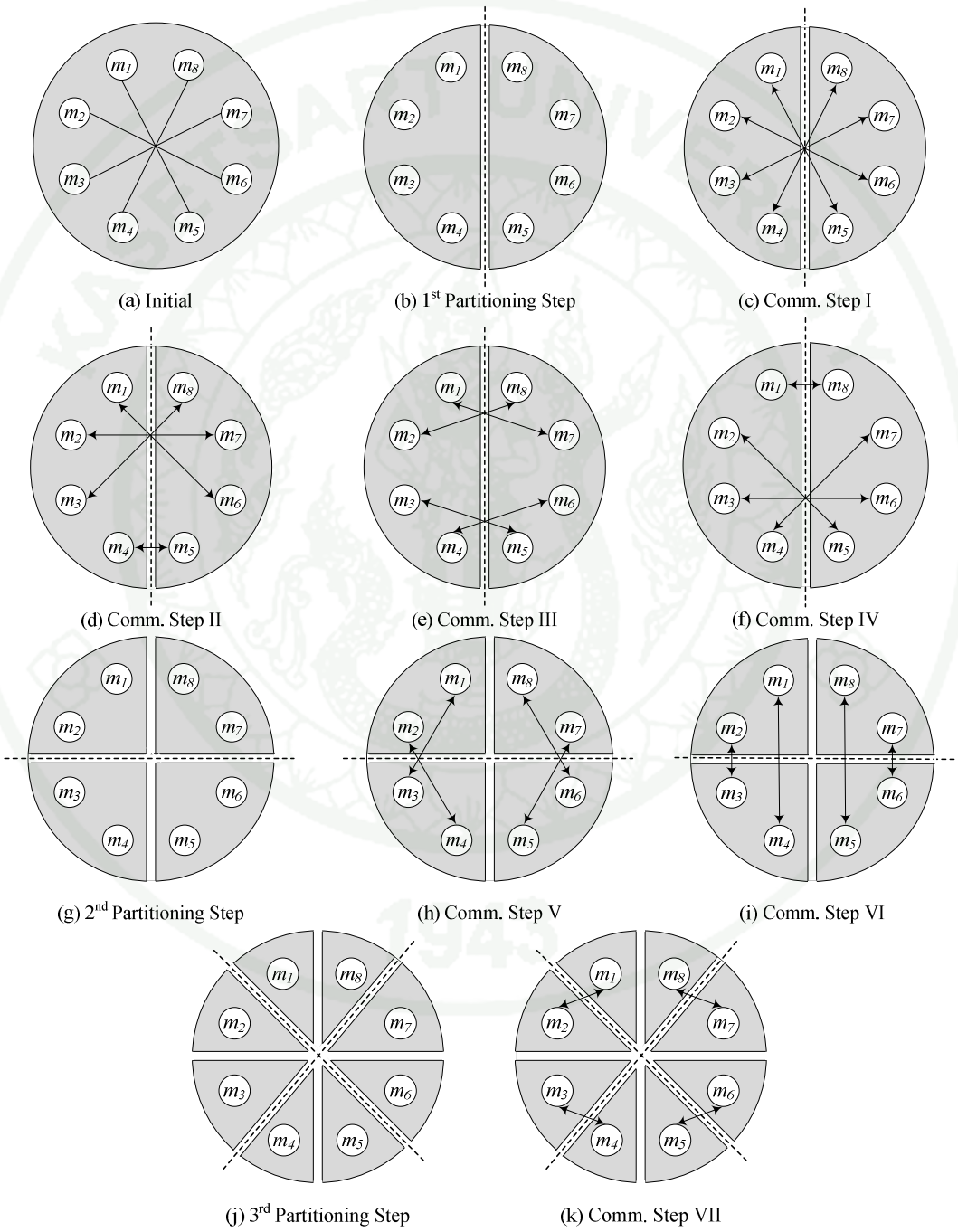

Figure 17 The partitioning and communication step of scheduling strategy of eight nodes.

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)*8 =56*. The example of results of bandwidth scheduling strategy of eight nodes is given in Figure 18.

1st Partitioning step									Comm. step
2 partitions:	$\{m_1$	m_2	m_3	$m_4\}$	$\{m_5$	m_6	m_7	$m_8\}$	
Sender	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	
Receiver	m_5	m_6	m_7	m_8	m_1	m_2	m_3	m_4	I
	m_6	m_7	m_8	m_5	m_2	m_3	m_4	m_1	II
	m_7	m_8	m_5	m_6	m_3	m_4	m_1	m_2	III
	m_8	m_5	m_6	m_7	m_4	m_1	m_2	m_3	IV
2nd Partitioning step									
4 partitions:	$\{m_1$	$m_2\}$	$\{m_3$	$m_4\}$	$\{m_5$	$m_6\}$	$\{m_7$	$m_8\}$	
Sender	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	
Receiver	m_3	m_4	m_1	m_2	m_7	m_8	m_5	m_6	V
	m_4	m_3	m_2	m_1	m_8	m_7	m_6	m_5	VI
3rd Partitioning step									
8 partitions:	$\{m_1\}$	$\{m_2\}$	$\{m_3\}$	$\{m_4\}$	$\{m_5\}$	$\{m_6\}$	$\{m_7\}$	$\{m_8\}$	
Sender	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	
Receiver	m_2	m_1	m_4	m_3	m_6	m_5	m_8	m_7	VII

Figure 18 The example of results of bandwidth scheduling strategy of eight nodes

RESULTS AND DISCUSSION

Results

1. Results of Improving I/O: Distributing Data and Scheduling I/O Operations

All the data distribution strategies and I/O scheduling algorithm in our simulator have been written in Java programming language and running under Microsoft Windows Vista. In all of the experiments, we fix the size of data-block to be 56 MB and I/O bandwidth to be 30 MBps. For our study, we investigate the impact of number of servers, number of clients, and number of data-block on the performance of different data distribution algorithms. To evaluate these data distribution strategies, we ran all of the considered strategies (Table 3) on variation system and the best strategy of each configuration as shown in Table 5 and the result of each experiment as shown in Figure 19.

Table 5 System Variation Architecture

Experiment no.	Number of server	Number of client	Number of data-block	Best result
1	40	40	40	I, II, IV, V
2	40	40	160	I, V
3	20	40	160	I, V
4	15	40	160	I, II, V

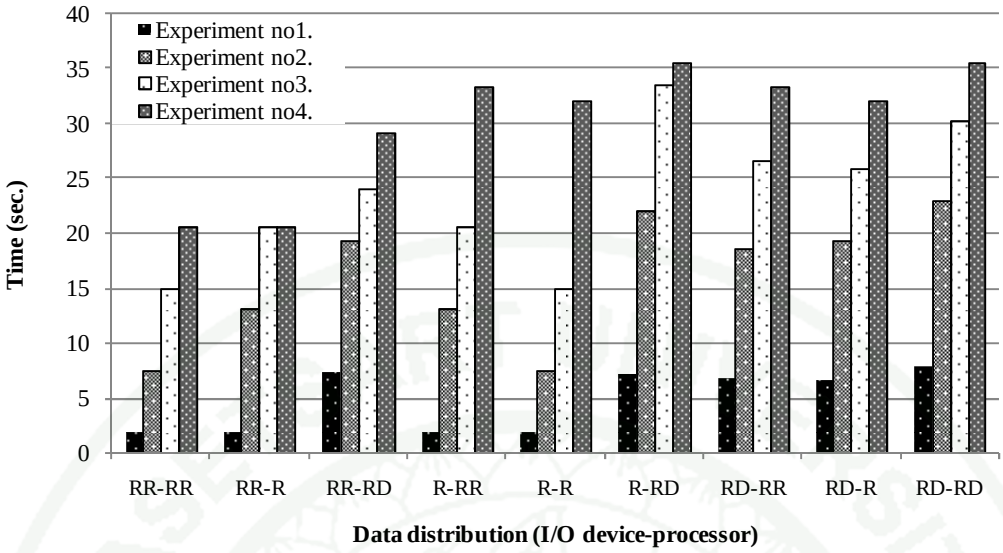

Figure 19 I/O time of vary data distribution of experimental setup no. 1, 2, 3, and 4

2. Results of Execution Scheduling of Task on Processors

In order to generate the test task graph, we mined a 60 million transaction database using the parallel FI-growth program (Manaskasemsak *et al.*, 2007). We utilized the standard "IBM synthetic data generator" (Srikant, 2006) to synthesize a transaction database. We used 1000 unique items to create 60 million records; each has average transaction length of 10. After the test graph is generated, we can run a scheduling simulator on this task graph using different scheduling strategies in Table 4. To evaluate these scheduling strategies, we ran all of the considered strategies on variation architecture as shown in Table 6.

Table 6 Multi-core variation architecture

	Total number of cores															
	8				16				32				64			
No. of machines	1	2	4	8	1	2	4	8	2	4	8	16	4	8	16	32
No. of cores per machine	8	4	2	1	16	8	4	2	16	8	4	2	16	8	4	2

The results show that various strategies can result in different level of performance on as scalable multi-core cluster systems that scale from 8-64 cores shown in Figure 20. Figure 21 shows processor workload on 32 core architecture that consists of 2 machines, 16 cores per machine.

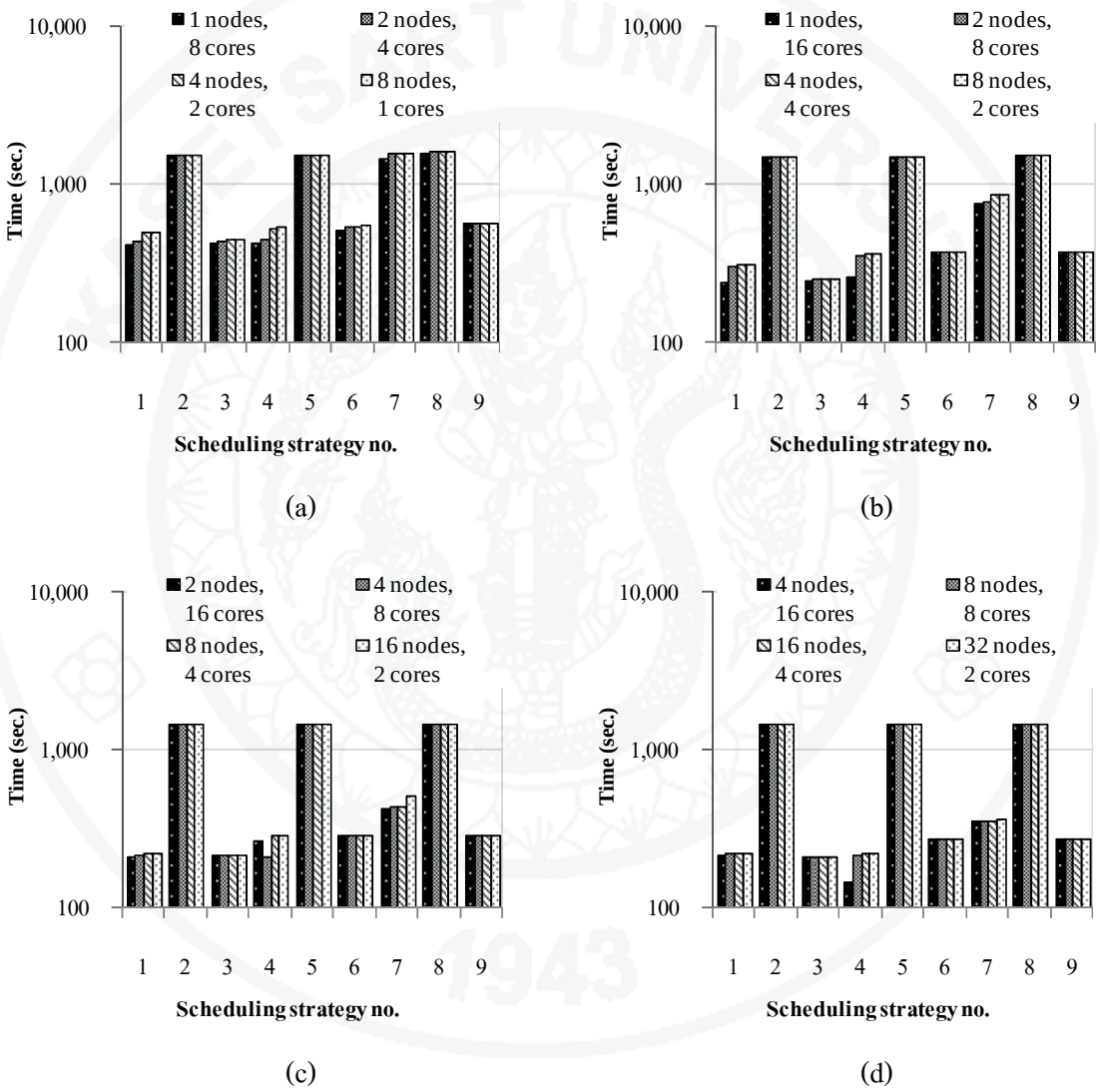

Figure 20 Run time using (a) 8 cores, (b) 16 cores, (c) 32 cores, and (d) 64 cores on variation architectures

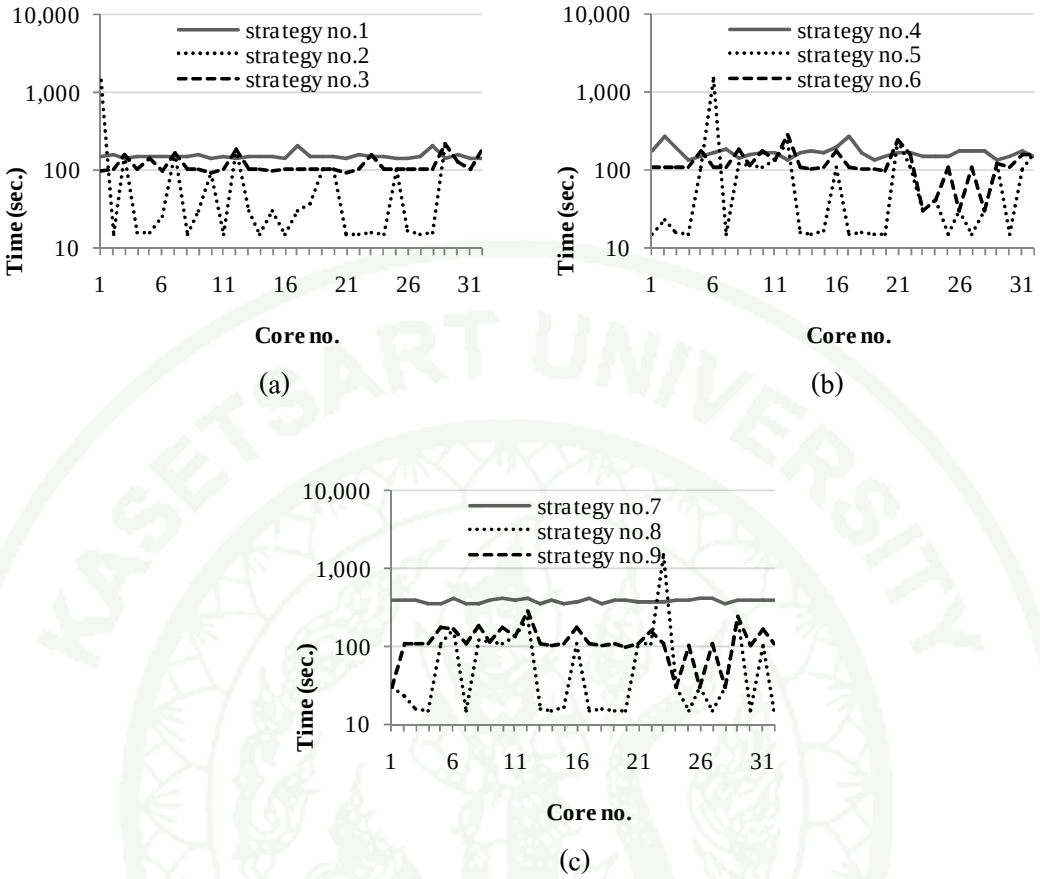

Figure 21 Workload of parallel FI-growth of the scheduling variation strategies on 32 cores (a) strategy no. 1, 2, and 3, (b) strategy no. 4, 5, and 6, and (c) strategy no. 7, 8, and 9

3. Results of Optimization of Communication Overhead

In this work, a database of 60 million transactions is mined by using the parallel FI-growth program (Manaskasemsak *et al.*, 2007) to generate the test task graph. We utilized the standard "IBM synthetic data generator" (Srikant, 2006) to synthesize a transaction database. We used 1000 unique items to create 60 million records, which have average transaction length of ten. The communication volume associated with each edge varies from 96 bytes to 768.8 K bytes. After the test graph is generated, we can run a scheduling simulator on this task graph using U-ICE scheduling strategy. To evaluate this scheduling strategy, we ran the considered strategy on variation architecture as shown in Table 7 and Table 8.

Table 7 Multi-core single system variation architecture

Multi-core Single system	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
CPU	Dual-core			Quad-core			Hexa-core			Octo-core		
No. of CPUs	1	2	4	1	2	4	1	2	4	1	2	4

Table 8 Multi-core cluster system variation architecture

Multi-core Cluster system	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20
CPU	Dual-core					Quad-core					Hexa-core					Octo-core				
No. of machines	2	4	8	16	32	2	4	8	16	32	2	4	8	16	32	2	4	8	16	32

We compare the result with scheduling strategy that without communication impact consideration, called Execution Only Scheduling (EO) that its method similar the best case of previous section. In EO method, the scheduling is divided into two steps; ordering of the tasks, and the mapping of the tasks onto processing units. In o*rdering step*, the task graph represented as a DAG is assigned a priority. It use smallest-numbered available task first or Left-to-Right (L-R) to determine the priorities of nodes. In m*apping step*, the mapping considers the factors of both the start-time of processor unit and size of data transfer from its parent nodes. 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.

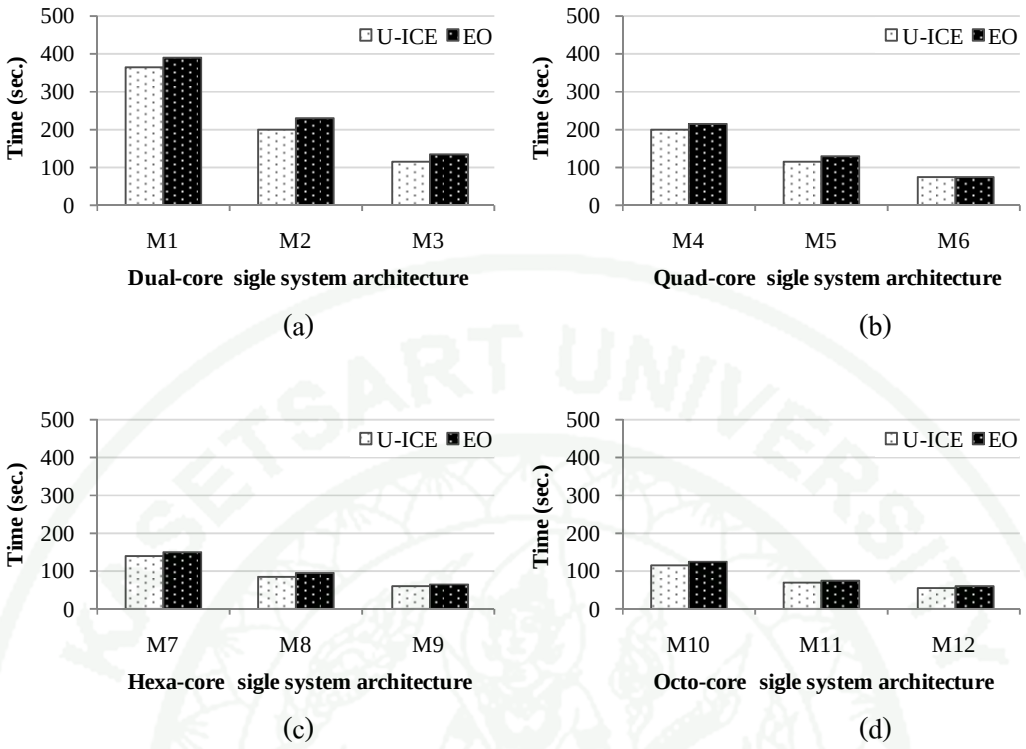

Figure 22 Runtime of parallel FI-growth using the U-ICE and EO scheduling strategies on (a) Dual-core, (b) Quad-core, (c) Hexa-core, (d) Octo-cores on multi-core single system variation architectures

Figure 22a, 22b, 22c, and 22d illustrate the runtime of parallel FI-growth using the U-ICE and EO scheduling strategies on dual-core, quad-core, hexa-core, and octo-core single system, respectively.

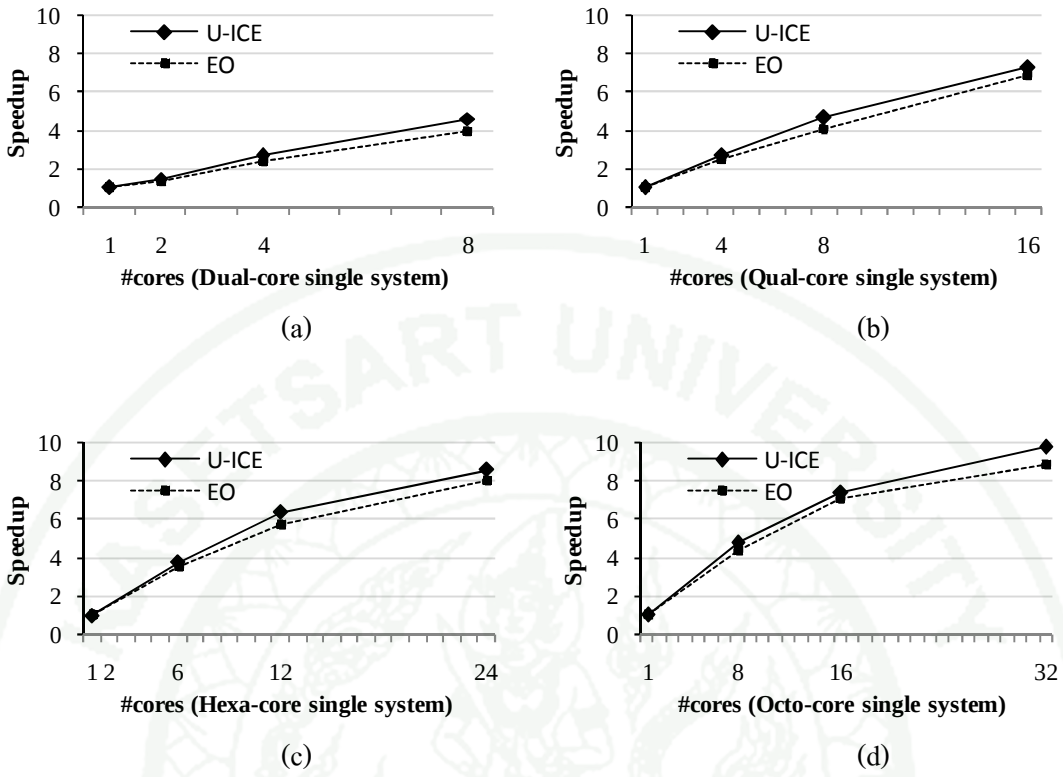

Figure 23 Speedup of parallel FI-growth using the U-ICE and EO scheduling strategies on (a) Dual-core, (b) Quad-core, (c) Hexa-core, (d) Octo-cores on multi-core single system variation architectures

Speedup of parallel FI-growth using the U-ICE and EO scheduling strategies on dual-core, quad-core, hexa-core, and octo-core single system as shown in Figure 23a, 23b, 23c, and 23d, respectively. The results clearly show that the U-ICE scheduling strategy gives a lower execution time than the EO scheduling strategy. In the best case, the communication time is reduced as much as 14.70% which run on a four dual-core CPU machine (M3 architecture).

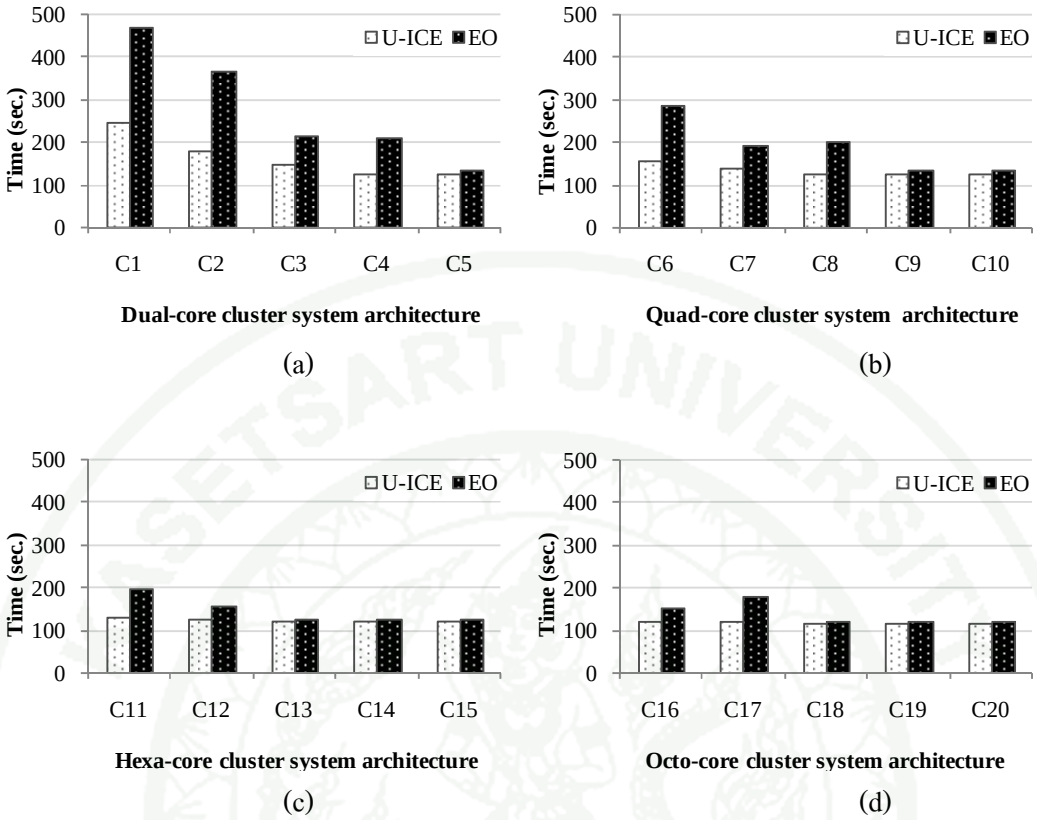

Figure 24 Runtime of parallel FI-growth using the U-ICE and EO scheduling strategies on (a) Dual-core, (b) Quad-core, (c) Hexa-core, (d) Octo-core on multi-core cluster variation architectures

Figure 24a, 24b, 24c, and 24d show runtimes of parallel FI-growth using the U-ICE and EO scheduling strategies on dual-core, quad-core, hexa-core, and octo-core multi-core cluster system, respectively.

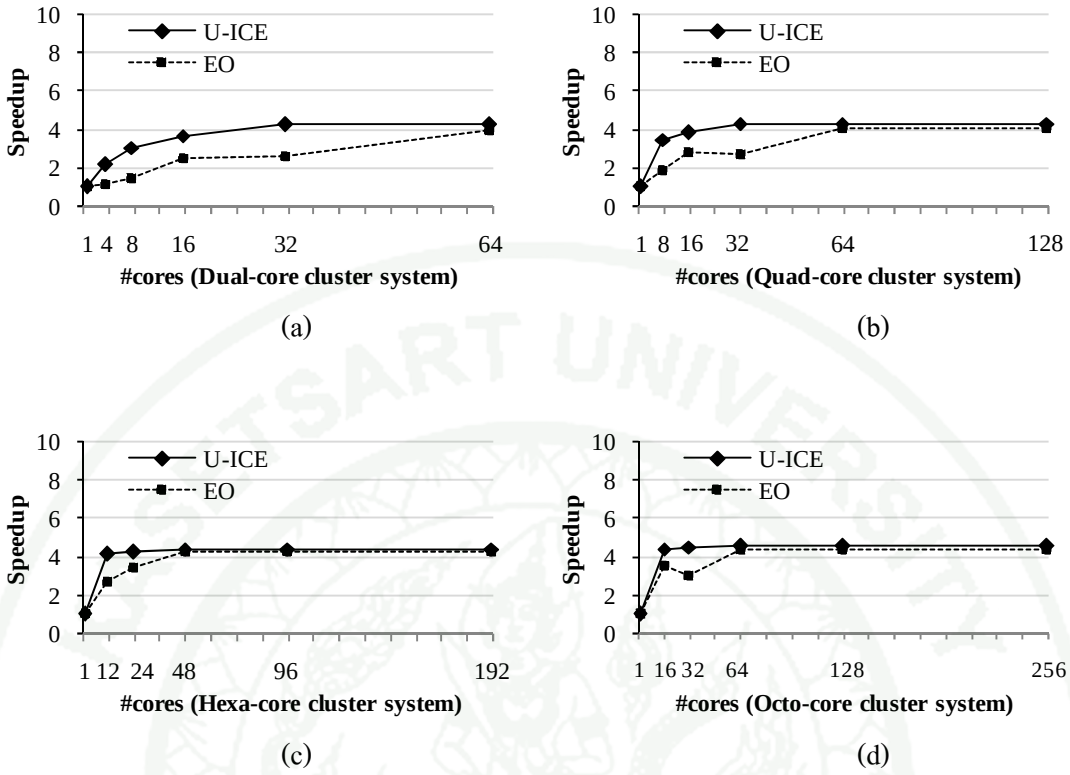

Figure 25 Speedup of parallel FI-growth using the U-ICE and EO scheduling strategies on (a) Dual-core, (b) Quad-core, (c) Hexa-core, (d) Octo-core on multi-core cluster variation architectures

Speedup of parallel FI-growth using the U-ICE and EO scheduling strategies on dual-core, quad-core, hexa-core, and octo-core cluster system as shown in Figure 25a, 25b, 25c, and 25d 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 total number of computing node is too much and the additional communication overhead incurred. For all variation multi-core cluster systems, the largest communication time is reduced as much as 51.45% and overall runtime is reduced as 42.35% which run on a cluster that consist of four dual-core machines (C2 architecture).

Discussion

1. Discussion of Improving I/O: Distributing Data and Scheduling I/O Operations

We have experimented with various data distributions on the large cluster simulator and have found them to be effective in improving performance. However, selecting a good data distribution requires the understanding about the parallel machine architecture and the data access patterns in the program.

As increasing amount of processor, I/O scheduling is an attempt to improve performance. The trade-off is that the execution time is lengthened because the total amount of communication increases with the number of processors.

2. Discussion of Execution Scheduling of Task on Processors

Figure 20a, 20b, 20c, and 20d illustrate the run times for nine scheduling strategies using 8, 16, 32, and 64 cores, respectively. The results show that run time of *Earliest start-time* strategy (scheduling strategies no.1 and 4) and *Largest size of data transfer & Earliest start-time* strategy (scheduling strategies no.3, 6, and 9) is lower than *Largest size of data transfer* strategy (scheduling strategies no. 2,5, and 8). Except run time of *Earliest start-time* strategy that makes a sequence by assigning *Top-to-bottom* priority is more than other scheduling strategies when decrease the amount of cores. The result is shown in Figure 20d. The execution time of best case is ten times less then worst case. For the ordering, the run time of *L-R* strategy (scheduling strategies no.1, 2, and 3 and *R-L* strategy (scheduling strategies no.4, 5, and 6) is similar and less than *T-B* strategy (scheduling strategies no.7, 8, and 9) especially when decrease amount of core.

For multi-core cluster system, load balancing among cores becomes a critical issue for high performance. Figure 21 shows processor workload on 32 core architecture that consists of 2 machines, 16 cores per machine. We can see that the scheduling strategy no.1, 4, and 7 uses

processor workload as the criterion of mapping tasks to cores. By contrast, the scheduling strategy no.2, 5 and 8 tries to minimize the communication costs between cores. As a result, the schedules generated by scheduling strategy no.1, 4, and 7 are well load balanced.

In conclusion, scheduling strategies no.1 and no.3 take time less than the other scheduling strategies and is well load balanced on small, medium, and large cluster configurations. For the architecture of cluster, that have total number of cores be equal but have different number of machines and number of core per machine, has a little affect when work with the same scheduling strategies.

3. Discussion of Optimization of Communication Overhead

We found that the runtime of U-ICE and EO on multi-core single machine consisting 32 cores (M12 architecture in Figure 22d) gives a lower runtime than the runtime of U-ICE and EO on multi-core cluster system in case of equal number of cores (C17 architecture in Figure 24d). This is caused by the problem workload is so small that it can be processed on a single system. In many organizations but the problem workload is so large that it cannot be solved on a single system that limits number of processor and memory. Clusters provide more computational resources that can solve this problem. We also found that the decreased communication time of U-ICE on multi-core cluster system was more than multi-core 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.

CONCLUSION

Now days, the big data has been created. This data come everywhere from social networking, mobile device applications, website server logs, RFID sensor event data, point-of-sale transactions and credit card purchases. However, raw data by itself does not provide much information. Association rule mining, a popular data mining technique, has been deployed in many decision support systems for years. It extracts significant information and then generates some interesting rules from large database. Association rule mining and big data make it possible for human societies to right decisions and bring enormous benefits to the social. Thus, a fast and scalable data mining technique becomes increasingly more important. Faster processing speed enables users to processes a much larger data set. Therefore, association rule mining applications can get benefits from the use of parallel computing systems to improve performance.

Although parallel association rule mining algorithms handles large datasets but faces challenges like data decomposition and I/O disk minimization, work load balancing and communication or synchronization minimization that needs attention to achieve high performance. Thus, this thesis proposes strategies to improve performance of a parallel association rule mining application on large cluster. A parallel FI-growth application is used to demonstrate the proposed concept and a multi-core cluster system that is a target system. This thesis addresses the three issues to support and speed up parallel association rule mining. First, this thesis proposes the improving I/O performance by selection proper data distribution and scheduling of parallel I/O data transfers on system. In this thesis, three simple data distribution strategies which consist of round robin, range, and random are evaluated with different number of data-blocks. We have experimented with various data distributions on the large cluster simulator and have found the round-robin data distribution strategy to be effective in improving performance. However, selecting a good data distribution requires the understanding about the parallel machine architecture and the data access patterns in the program.

Second, this thesis provides the scheduling strategy to efficiently perform the execution task of association rule mining. The designing of scheduling strategy for parallel FI-growth application, the execution is divided into two steps that consist of task ordering and task mapping. To ordering the task, a sequence of task for scheduling is created by assigning them priority. In this thesis, we use three ways to determine the priorities of nodes that consist of smallest-numbered available task first or Left-to-Right (L-R), largest-numbered available task first or Right-to-Left (R-L) and Top-to-bottom (T-B). For task mapping, we considered three criterions for mapping tasks to processing unit. First, tasks are allocated to a processing unit which allows the earliest start-time. Second, tasks are allocated by considering the size of data transfer from its parent. The task is allocated to a same processing unit of its parent node which has largest size of data transfer. Finally, Tasks are allocated by considering the factors of both the start-time of processor unit and size of data transfer from its parent nodes. At first we consider size of data transfer from its parent nodes and then follow by the earliest start-time. By combining these variations of ordering and mapping strategies, we obtain nine basic scheduling strategies. All of the scheduling strategies are run on variation architecture for evaluating the performance. The results show that scheduling strategy which use Left-to-Right way to determine the priorities of tasks and allocate task to processor by considering the factors of the start-time of processor unit take time less than the other scheduling strategies and is well load balanced on small, medium, and large cluster configurations.

Finally, this thesis proposes managing task of communication by using the proper scheduling of communication message that considers the right sequence of message transmission and interconnection network utilization. An approach called Unified I/O, Communication and Execution Scheduling (U-ICE) is proposed. The concept is to combine I/O, computation, and communication scheduling together to achieved a much better performance. The U-ICE scheduling, the execution is decomposed into two steps, the nodes partitioning and communication scheduling. These two steps execute alternately. In partitioning step, a set of compute nodes is divided into two balanced partitions and then recursively divided the two partitions until the number of node of each partition is one. In each partitioning step, all nodes in

each partition are scheduled to send/receive message to/from all nodes within another partition. 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. We run a scheduling simulator on generated task graph using U-ICE scheduling strategy. To evaluate this scheduling strategy, we ran considered strategy on multi-core single machine variation architecture and multi-core cluster system variation architecture. We compare the result with scheduling strategy that without communication impact consideration, called Execution Only (EO) Scheduling. The experimental results on multi-core single machine variation architecture, the results clearly show that the U-ICE scheduling strategy gives a lower execution time than the EO scheduling strategy. In the best case, the communication time is reduced as much as 14.70% which run on a four dual-core CPU machine. While, the communication times for U-ICE and EO scheduling strategies on various multi-core cluster systems, the results show that the largest communication time is reduced as much as 51.45% and overall runtime is reduced as 42.35% by using U-ICE scheduling which run on a cluster that consist of four dual-core machines. We also found that the decreased communication time of U-ICE on multi-core cluster system was more than multi-core single machine in case of equal number of cores. This is the result of a better utilization of interconnection network used to build the system.

We propose that the use of good scheduling strategy can helps speeding up the execution of parallel FI-growth data mining algorithm. We propose a simple step in generating the strategies and then building a tool to evaluate the merit of each strategy on a real task graph obtained from the application on a target multi-core cluster system. Currently, we are looking forward to applying this approach to other algorithm and other hardware platforms. We hope that this work can lead to the development of data mining algorithm that can handle a massive scale data faster and lead us to discover more knowledge that is useful for us.

LITERATURE CITED

Agrawal, R., T. Imielinski and A. Swami. 1993. Mining Association Rules between Sets of Items in Large Databases, pp. 207-216. *In* **Proceedings of the ACM SIGMOD Conference on Management of Data.** Washington, D.C., United States.

_____ and J.C. Shafer. 1996. Parallel Mining of Association Rules. **IEEE Transactions on Knowledge and Data Engineering** 8 (6): 962-969.

_____ and R. Srikant. 1994. Fast Algorithms for Mining Association Rules, pp. 487-499. *In* **Proceedings of 20th International Conference on Very Large Data Bases, VLDB'94.** Morgan Kaufmann, Santiago de Chile, Chile.

Agarwal, R.C., C.C. Aggarwal and V.V.V. Prasad. 2001. A Tree Projection Algorithm for Generation of Frequent Item Sets. **Journal of Parallel and Distributed Computing** 61 (3): 350-371.

Amphawan, K. and A. Surarerks. 2005. An Approach of Frequent Item Tree for Association Generation, *In* **Proceedings of the IASTED Conference on Artificial Intelligence and Soft Computing 2005, ASC2005.** Benidorm, Spain.

Buehrer, G., S. Parthasarathy, S. Tatikonda, T. Kurc and J. Saltz. 2007. Toward Terabyte Pattern Mining: An Architecture-Conscious Solution, pp. 2-12. *In* **Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming.** ACM, San Jose, California, USA.

Dehao, C., L. Chunrong, H. Wei, C. WenGuang, Z. Yimin and Z. Weimin. 2006. Tree Partition Based Parallel Frequent Pattern Mining on Shared Memory Systems, pp. 363-370. *In* **20th Int. Parallel and Distributed Processing Symposium, 2006 (IPDPS 2006).**

Gantz, J. and D. Reinse. 2011. **The 2011 Digital Universe Study: Extracting Value from Chaos**. Available Source: http://www.emc.com/digital_universe, June 2011.

Han, E.H., G. Karypis and V. Kumar. 1997. Scalable Parallel Data Mining for Association Rules, pp. 277-288. *In* **Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data.** ACM Press, Tucson, Arizona, United States.

Han, J., J. Pei and Y. Yin. 2000. Mining Frequent Patterns without Candidate Generation, pp. 1-12. *In* **Proceedings of the 2000 ACM SIGMOD International Conference on Management of data.** ACM Press, Dallas, Texas, United States.

Javed, A. and A. Khokhar. 2004. Frequent Pattern Mining on Message Passing Multiprocessor Systems. **Distributed and Parallel Databases** 16 (3): 321-334.

Jianyong, W., J. Han, Y. Lu and P. Tzvetkov. 2005. TFP: an efficient algorithm for mining top-k frequent closed itemsets. **IEEE Transactions on Knowledge and Data Engineering** 17(5): 652-663.

Kun-Ming, Y. 2011. An Efficient Load Balancing Multi-Core Frequent Patterns Mining Algorithm, pp. 1408-1412. *In* **2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications.**

Lei, C., G. Qi and D.K. Panda. 2007. Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System, pp. 471-478. *In* **Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007.**

Lin, K.-C., I.-E. Liao and Z.-S. Chen. 2011. An Improved Frequent Pattern Growth Method for Mining Association Rules. **An International Journal Expert Systems with Applications** 38 (5): 5154-5161.

Manaskasemsak, B., N. Benjamas, A. Rungsawang, A. Surarerks and P. Uthayopas. 2007. Parallel Association Rule Mining Based on Fi-Growth Algorithm, pp. 1-8. *In* **Proc. of the 2007 International Conference on Parallel and Distributed Systems.**

Pan, F., G. Cong, A.K.H. Tung, J. Yang and M.J. Zaki. 2003. Carpenter: Finding Closed Patterns in Long Biological Datasets, pp. 637-642. *In* **Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.** ACM, Washington, D.C.

Park, J.S., M. Chen and P.S. Yu. 1995a. An Effective Hash-Based Algorithm for Mining Association Rules, pp. 175-186. *In* **Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data.** ACM, San Jose, California, United States.

_____, _____ and _____. 1995b. Efficient Parallel Data Mining for Association Rules, pp. 31-36. *In* **Proceedings of the fourth international conference on Information and knowledge management.** ACM, Baltimore, Maryland, United States.

Parthasarathy, S., M.J. Zaki, M. Ogihara and W. Li. 2001. Parallel Data Mining for Association Rules on Shared Memory Systems. **Knowledge and Information Systems** 3 (1): 1-29.

Pasquier, N., Y. Bastide, R. Taouil and L. Lakhal. 1999. Discovering Frequent Closed Itemsets for Association Rules, pp. 398-416. *In* **Proceedings of the 7th International Conference on Database Theory.** Springer-Verlag,

Savasere, A., E. Omiecinski and S.B. Navathe. 1995. An Efficient Algorithm for Mining Association Rules in Large Databases, pp. 432-444. *In* **Proceedings of the 21th International Conference on Very Large Data Bases.** Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Skillicorn, D. 1999. Strategies for Parallel Data Mining. **IEEE Concurrency** 7 (4): 26-35.

Srikant, R. 2006. **Synthetic Data Generation Code for Associations and Sequential Patterns**. Available Source: http://www.almaden.ibm.com/cs/disciplines/iis/,

Tanbeer, S.K., C.F. Ahmed and J. Byeong-Soo. 2009. Parallel and Distributed Frequent Pattern Mining in Large Databases, pp. 407-414. *In* **Proc. of the 11th IEEE International Conference on High Performance Computing and Communications, HPCC '09.**

Wang, J., J. Han and J. Pei. 2003. Closet+: Searching for the Best Strategies for Mining Frequent Closed Itemsets, pp. 236-245. *In* **Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.** ACM, Washington, D.C.

Yu, K.-M., J. Zhou and W. Hsiao. 2007. Load Balancing Approach Parallel Algorithm for Frequent Pattern Mining. **Parallel Computing Technologies** 4671: 623-631.

Zaiane, O.R., M. El-Hajj and P. Lu. 2001. Fast Parallel Association Rule Mining without Candidacy Generation, pp. 665-668. *In* **Proceedings of the IEEE International Conference on Data Mining, 2001.**

Zaki, M.J. and C.J. Hsiao. 2002. Charm: An Efficient Algorithm for Closed Itemset Mining, pp. 457-473. *In* **Proceedings of the 2002 SIAM International Conference on Data Mining. (SDM '02).**

Zaki, M.J. and C.J. Hsiao. 2005. Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure. **IEEE Transactions on Knowledge and Data Engineering** 17 (4): 462-478.

_____, M. Ogihara, S. Parthasarathy and W. Li. 1996. Parallel Data Mining for Association Rules on Shared-Memory Multi-Processors, pp. 34-43. *In* **Proceedings of the 1996 ACM/IEEE Conference on Supercomputing, 1996.**

CURRICULUM VITAE

NAME : Ms. Nunnapus Benjamas

BIRTH DATE : June 30, 1978

BIRTH PLACE : Phetburi, Thailand

EDUCATION :

YEAR	INSTITUTE	DEGREE/DIPLOMA
2000	KhonKaen Univ.	B.Sc. (Computer Science)
2005	Kasetsart Univ.	M.S. (Computer Science)

SCHOLARSHIP/AWARDS : Higher Education Commission, Ministry of Education, Thailand 2005-2007