Massively Parallel Distributed Feature Extraction in Textual Data Mining Using HDDI™

Jirada Kuntraruk and William M. Pottenger
Computer Science and Engineering Department
Lehigh University
Bethlehem, PA 18015 USA
{jak5,billp}@cse.lehigh.edu

Abstract

One of the primary tasks in mining distributed textual data is feature extraction. The widespread digitization of information has created a wealth of data that requires novel approaches to feature extraction in a distributed environment. We propose a massively parallel model for feature extraction that employs unused cycles on networks of PCs/workstations in a highly distributed environment. We have developed an analytical model of the time and communication complexity of the feature extraction process in this environment based on feature extraction algorithms developed in our textual data mining research with HDDI™ [1] [18] [20]. We show that speedups linear in the number of processors are achievable for applications involving reduction operations based on a novel, parallel pipelined model of execution. We are in the process of validating our analytical model with empirical observations based on the extraction of features from a large number of pages on the World Wide Web.

1 Introduction

Recent advances in computer technology are fueling radical changes in the nature of information management. Increasing computational capacities coupled with the ubiquity of networking have resulted in widespread digitization of information, thereby creating fundamentally new possibilities for managing information. One such opportunity lies in the budding area of textual data mining. With roots in the fields of statistics, machine learning and information theory, data mining is emerging as a field of study in its own right. The marriage of data mining techniques to applications in textual information management has created unprecedented opportunity for the development of automatic approaches to tasks heretofore considered intractable.

One of the primary tasks in mining distributed textual data is feature extraction. As noted, the widespread digitization of information has created a wealth of data that requires novel approaches to feature extraction in a distributed environment. Our approach incorporates the development of a massively parallel model for feature extraction that employs unused cycles on networks of PCs/workstations in a highly distributed environment. The model has two components: an analytical component and an empirical component. The analytical component models the time and communication complexity of the feature extraction process in a massively parallel distributed environment. Based on feature extraction algorithms developed in our textual data mining research with HDDI™ [1] [18] [20], we are also performing empirical validation of the analytical model.

In the following sections we first discuss the related work. We outline the feature extraction process based on our previous work in [1], and then develop our analytical model. Validation of the model is discussed in Section 6.

2 Related Work

Massively parallel and distributed processing is widely recognized as a key technology of the future. The increase in microprocessor performance coupled with greater memory capacities and network bandwidth are indicative of the fact that massively parallel, distributed systems are an attractive alternative to supercomputers in terms of both price and performance for many applications. One of the outcomes of research in such scalable systems has been the development of various models of performance.

For example, [9], [13] and [15] discuss various aspects of speedup and efficiency of parallel systems. Although insightful, these do not provide explicit analytical models. One goal of our research is to identify an analytical model that predicts the execution time of an application given a set of parameters for a massively parallel distributed frame-

work.

There are two important factors that dominate the execution time in parallel and distributed processing: the computation time and the communication time. [6], [8] and [16] present communication models for various distributed-memory architectures. Of these, we have chosen to base our analytical model on [8]. We discuss this model further in Section 4.

[10] and [11] propose models that predict the execution time of an application that incorporates both the computation time and the communication time. However, [10] does not model the communication in sufficient detail. [11] has a reasonable model for the communication, but bases the computational model on the average execution time of the processors in the system. It is our contention that a more accurate model of computation time would measure the execution time of a parallel distributed application by the wall-clock time from the start of computation until the last processor to finish a task has done so. As a result, the computational model presented in [11] was deemed less than optimal and we decided to develop our own model. This model is presented in Section 4.

3 Feature Extraction in HDDI™

In this section we review the three functional parts of the HDDI™ feature extraction process: input, part of speech tagging, and concept extraction.

3.1 Terminology

Before we introduce the functional parts of the system, we must introduce some terminology:

- Items: Item refers to the basic unit of data content that is used in textual data mining. We generally use item to refer to a single document; however as explained in [2], other units of information, such as subsections of a document or sentences, could be used as well. For simplicity we will assume that item refers to a document in this article.

- Collections: A collection refers to a group of items that will be indexed in the HDDI™ textual data mining system.

- Concepts: We use concept to refer to a maximal length English-language noun phrase that is extracted from a collection's items. Concepts are extracted as features and used as keywords in the HDDI™ textual data mining system.

3.2 Input

Since a collection can originate from any source, we need to handle different input formats including SGML and various subsets such as HTML and XML. In addition, the feature extraction process requires us to identify particular fields of data in the input collection that are of interest (e.g., the title of an item). In order to accomplish these tasks we developed an extensible, reusable object-oriented input parser. See [1] for details.

3.3 Part of Speech Tagging

After identifying fields of interest, our feature extraction algorithms perform part of speech tagging. The part of speech tagger is a rule-based system for tagging English parts of speech. This system is based on the SemanTag system developed in [7], which in turn is based on [3] [4] [5]. The tagger uses three levels of rule sets to determine the part of speech of each word, and tags words with their English part of speech tag, as specified in the Brown tagset [12].

DT - determiner
IN - preposition or subordinating conjunction
NN - noun - singular or mass
PP - personal pronoun
VBD - verb - past tense
. - literal period

Figure 1. Selected Brown part of speech tags and their definitions

3.4 Feature Extraction

A key part of textual data mining is feature, or concept extraction. For this purpose, we have designed and implemented a sophisticated English language noun phrase extractor. Our premise is that maximal length noun phrases are high quality discriminators and should therefore be used as keyword features for indexing purposes by the HDDI™ textual data mining system. In order to identify maximal length noun phrases from the tagged text, a finite state machine capable of handling complex noun phrases was generated [1].

Concurrently with the extraction of noun phrases, other information that is used later in the HDDI™ model building stage is extracted and preserved. For example, a frequency of occurrence is calculated for each concept in each item as well as the character offset of each concept in the original item. Also, the field in which the concept occurred (e.g., title) is preserved.

The following is an example of the functionality of the feature extractor. If the extractor received the input "She built an apparatus for the transformation of picture information." as the original text of an item, and "She//PP built//VBD an//DT apparatus//NN for//IN the//DT transformation//NN of//IN picture//NN information//NN .//." as the marked up text (see Figure 1 for an explanation of the part of speech tags), the noun phrase "apparatus for the transformation of picture information" would be extracted with a character offset of 14. This concept would be given a frequency of occurrence of one since it occurs only once in this simple one sentence example. Features of this nature are useful in a variety of textual data mining tasks [18].

4 Analytical Model

To adapt the feature extraction algorithms to a distributed environment, we added an initial retrieval step. Given a list of URLs as input, feature extraction consists of the following four functional parts: URL content retrieval, input, part of speech tagging and concept extraction.

4.1 Computational Model

The approximate time and space complexity of each of the four modules in the feature extraction process is shown in Table 1. The computational model incorporates the following parameters:

- m is the average size of the content of the input URLs,
- L is the size of the Lexicon,
- LR is the number of Lexical rules,
- LRF is the size of the Lexical rules file,
- CR is the number of Contextual rules,
- CRF is the size of the Contextual rules file.

Module	Runtime	Space
Retrieval	$O(m)$	$O(m)$
Input	$O(m)$	$O(m)$
Tagging	$O(m \cdot (LR + CR))$	$O(m + L + LRF + CRF)$
Concept Extraction	$O(m)$	$O(m)$

Table 1. Time and Space Complexity of the HDDI™ Feature Extraction Modules

These four modules are packed into a monatomic computational task. The computation time can be expressed as:

$$T_{Comp} = T_{Ret} + T_{Inp} + T_{Tag} + T_{Ext} \quad (1)$$

Where

- T_{Ret} is the time for the retrieval module,
- T_{Inp} is the time for the input module,
- T_{Tag} is the time for the tagging module,
- T_{Ext} is the time for the concept extraction module.

4.2 Communication Model

We model the communication using the LogP model [8]. LogP is a model of a distributed-memory multiprocessor. The processors communicate by point-to-point messages. There are four main parameters of the model:

P: the number of processors.
L: an upper bound on the latency or the time that the message spends in the interconnection network.
o: the overhead or the time for the processor to inject the message into or pull the message from the interconnection network.
g: the gap or the minimum time interval between consecutive message transmissions or receptions at a processor. It is a reciprocal of available bandwidth of the interconnection network.

We choose to use the LogP model because the model specifies the performance characteristics of the interconnection network but does not describe the structure of the network, i.e., the model accounts for communication costs without assuming a topology of the interconnection network. Therefore, communication models based on LogP are portable from one platform to another.

Using the LogP model, the time to deliver a message can be expressed algebraically as

$$T_{Comm} = 2o + L + (Msg\ size) \cdot g \quad (2)$$

This equation is similar to the equation presented in [16] which is

$$T_{Comm} = 2o + L + \frac{Msg\ size}{Bandwidth} \quad (3)$$

In this analysis we assume that $o \leq g$. Then we have

$$T_{Comm} = L + (Msg\ size) \cdot g \quad (4)$$

We replace $Msg\ size = k \cdot i$ when we assume that the average number of characters in a feature is i, and k is the average number of features extracted from a URL of size m. Then

$$T_{Comm} = L + (k \cdot i) \cdot g \quad (5)$$

4.3 Parallel Pipelined Reduction Model

The ability to perform feature extraction in a parallel, distributed environment relies on the fact that feature extraction is an associative operation and can, as a result, be parallelized [19]. Note that feature extraction, like all associative operations, requires that a reduction be performed. Computationally, feature extraction can be modeled as two different tasks: first, the monatomic computational task noted previously in Section 3, and second, a parallel merge as discussed in [17]. Of these two tasks, the parallel merge forms the reduction stage of the computation. In a distributed environment, communication takes place during the reduction. This communication is represented by the arrows in an example reduction pictured in Figure 2. The complexity of each step is modeled as *cost = computation time + communication time*, where the cost is dominated by the computation time of the monatomic feature extraction task. We discuss this constraint on the model in what follows.

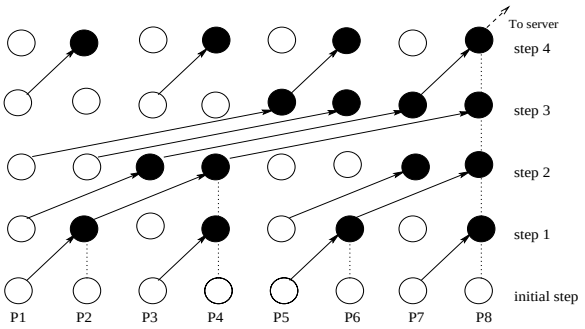

Figure 2. Reduction model. The white nodes represent the operation of feature extraction and the black nodes represent the merging operation. This figure depicts execution on eight processors. The arrow edges represent the communication that takes place. The dotted lines and the arrow edges together form a reduction tree.

Figure 2 depicts an example of our reduction model on eight processors. During the initial step (step 0) every processor executes the monatomic feature extraction task. Then, starting with step one, a reduction is completed every $\lg P$ steps[1]. The system reaches a state of equilibrium after $\lg P$ steps at which time $\frac{P}{2}$ processors continuously perform feature extraction and $\frac{P}{2}$ processors perform merging. This forms, in essence, a pipelined, parallel reduction consisting of $2 * \lg P + 1$ stages in which new content is continually being processed in the feature extraction task, and pipelined to the $2 * \lg P$ stages of the reduction tree[2]. The lengths of the $2 * \lg P + 1$ stages in the pipeline are constrained such that all stages are equal in length, thus guaranteeing the optimality of the pipelined reduction [16].

4.3.1 The Merge

The implementation of distributed HDDI™ feature extraction incorporates a binary tree data structure. The process of feature extraction stores each feature present in the input in lexicographical order in this data structure. At each node of the reduction tree two such binary trees of features are merged. The time complexity of the merge operation for two binary trees is $O(k \cdot \lg l)$, where k and l are the number of features of the two trees being merged. With no loss in generality, we assume that $l \geq k$. Initially, we make the assumption that there is no overlap of features (i.e., no noun phrase occurs more than once). Because of this, our complexity analysis will result in an upper bound for the merge time.

Figure 2 depicts the merging operation for an example of eight processors. Starting with the initial step, each processor creates a tree of features. The features from P_1 are then sent to P_2 to be merged with the features extracted by P_2. Likewise, P_3 sends its features to P_4 and so on as portrayed by the arrow edges in Figure 2. Thus merging takes place on the processors that receive the data. This operation continues until the binary feature trees have all been merged (on P_8 in Figure 2 by Step 3). The dotted lines in Figure 2 aid in the visualization of the reduction.

Our reduction model employs a complete binary tree. This is a data structure used in several known algorithms for reductions (e.g., prefix summing [14] [21]). The use of a complete tree is in this sense an optimal component of the model.

4.4 Model Complexity and Optimality

Figure 3 depicts the same eight processors as Figure 2 with the number of messages being transmitted explicitly noted on the communication paths[3]. Messages consist of binary trees of features as discussed previously. As noted, we assume that the number of features grows by a factor of

[1] We make the simplifying assumption that the number of processors P is a power of 2. Note that $\lg x = \log_2 x$.

[2] Note that unlike a hardware pipeline, the communication between stages in the reduction tree is significant and as a result is modeled as $\lg P$ of the $2 * \lg P + 1$ stages.

[3] Again, for clarity in understanding the reduction tree, dotted lines are added that record the number of messages currently retained in a given node.

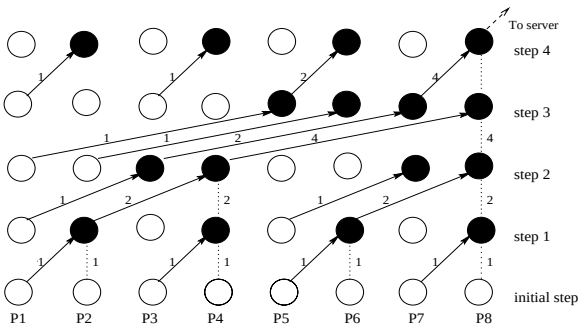

Figure 3. Reduction Tree. The dotted lines and the arrow edges form a binary reduction tree. The weight of each arrow edge represents the number of messages passing through the communication channel.

two. This is a conservative estimate assuming no overlap in feature space, and as such yields an upper bound on the actual communication complexity.

At step j, there are $P + j \cdot \frac{P}{2}$ URLs retrieved. It will take another $\lg P$ additional steps to collect all the features extracted from the content of these URLs into the central repository. Therefore, the total time to process $P + j \cdot \frac{P}{2}$ URLs is computed as:

$$\begin{aligned} T_{Total} &= (j + \lg P) \cdot (T_{Comp} + T_{Comm}) \\ &= (j + \lg P) \cdot (T_{Comp}) + \\ &\quad [\{L + (k \cdot i \cdot g)\} + \{L + 2 \cdot (k \cdot i \cdot g)\} + \\ &\quad \cdots + \{L + 2^{\lg P - 1} \cdot (k \cdot i \cdot g)\} + \\ &\quad (j-1) \cdot \{L + \frac{P}{2} \cdot (k \cdot i \cdot g)\}] + \\ &\quad L + P \cdot (k \cdot i \cdot g) \\ &= (j + \lg P) \cdot (T_{Comp}) + (j + \lg P) \cdot L + \\ &\quad (k \cdot i \cdot g) \cdot \sum_{n=0}^{\lg P - 1} 2^i + \\ &\quad (j-1) \cdot \frac{P}{2} \cdot (k \cdot i \cdot g) + P \cdot (k \cdot i \cdot g) \end{aligned} \tag{6}$$

In the derivation above we have replaced T_{Comm} with equation (5) and then multiplied the result by $j + \lg P$, the number of times communication takes place. As we mentioned at the beginning of this section, the number of features k grows by a factor of two until it reaches the bound of $O(\frac{P}{2})$. This results in an proportional increase in communications complexity. T_{Comp} can be replaced by equation (1).

4.4.1 The Optimality

Due to the nature of the binary reduction tree, message size reaches a bound of $O(\frac{P}{2})$ when the computation reaches step $\lg P$. After step $\lg P$, message size remains constant. As noted previously, the system reaches a state of equilibrium that optimally uses the processors and communication resources given the constraints of the feature extraction algorithm. This optimal use of resources depends on the $2 * \lg P + 1$ stages being equal in length. These stages consist of T_{Comp}, $(\lg P - 1) * T_{Merge}$, $\lg P * T_{Comm}$ and $T_{CommServer}$ as depicted in Figure 4. From the model we know that T_{Comm} and $T_{CommServer}$ are bounded by $L + P \cdot (k \cdot i \cdot g)$. Thus, if we constrain $T_{Merge} \leq T_{Comp}$, the stages will be of equal length given that the following is satisfied:

$$T_{Comp} \geq L + P \cdot (k \cdot i \cdot g) \tag{7}$$

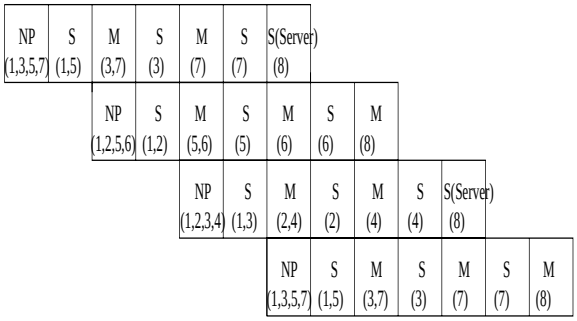

Figure 4. Parallel Pipeline. The parallel reduction pipeline of seven stages used in execution on eight processors

4.5 The Speedup Model

The speedup model incorporates speedups due to both parallel and pipelined execution as depicted in Figure 4 for an eight processor example. Assume that N is the number of items (pages) and P is the number of processors. The sequential execution time is

$$T_{Seq} = N \cdot T_{Comp} + (N-1) \cdot T_{Merge} \tag{8}$$

Where T_{Comp} is the time to perform feature extraction on one item and T_{Merge} is the merge time for adding the extracted features from a new item to the existing list of features.

The parallel execution time for one item-set of $\frac{P}{2}$ items is

$$\begin{aligned} T_{Par} = \frac{N}{\frac{P}{2}} \cdot T_{Comp} + \frac{N}{\frac{P}{2}} \cdot \\ (T_{Merge} + T_{Comm}) \cdot \lg P \end{aligned} \tag{9}$$

Here T_{Comp} is the feature extraction time for one item, T_{Merge} is an upper bound on the merge time, and T_{Comm} is an upper bound on the communication time for one or more items. The first term, $\frac{N}{\frac{P}{2}} \cdot T_{Comp}$, represents the execution time to extract features from N items using $\frac{P}{2}$ processors. The second term, $\frac{N}{\frac{P}{2}} \cdot (T_{Merge} + T_{Comm}) \cdot \lg P$, represents the reduction (combination) of the $\frac{N}{\frac{P}{2}}$ item-sets. Each reduction of $\frac{P}{2}$ items in a set takes $\lg P$ merges and $\lg P$ communications on a single set of $\frac{P}{2}$ processors, so the total reduction time for each set of $\frac{P}{2}$ items is $(T_{Merge} + T_{Comm}) \cdot \lg P$.

Generalizing from Figure 4 we have

$$Pipeline\ depth = 2 \cdot \lg P + 1 \tag{10}$$

This derives directly from the model. However, the actual maximum theoretical speedup is $2 \cdot \lg P$ due to a functional hazard in the first two stages of the pipeline (for example in Figure 4 processors 1,3,5,7 perform feature extraction in pipeline stage one, then 1,5 send to 3,7, so none of these four processors are free until the end of stage two and no other processors are available because they are being use in other (e.g., reduction) operations when the pipeline is full).

The speedup due to parallel execution of one item-set on a single set of $\frac{P}{2}$ processors is

$$\begin{aligned} S_{Par} &= \frac{T_{Seq}}{T_{Par}} \\ &= \frac{P}{2 \cdot \lg P + 1} \end{aligned} \tag{11}$$

This assumes that $T_{Comp} \approx T_{Merge}$ during sequential execution and that $T_{Comp} \approx T_{Merge} \approx T_{Comm}$ during parallel execution (i.e., all pipeline stages are approximately equal[4].

Our model assumes continuous, never-ending operation of the feature extraction process in an application such as updating a large search engine database (e.g., www.google.com). We thus assume that the overall speedup is the product of the speedup resulting from parallel extraction of features from a group of $\frac{P}{2}$ items (i.e., $\frac{T_{Seq}}{T_{Par}}$ above) multiplied by the depth of the pipeline. The overall speedup is thus

$$\begin{aligned} S_{overall} &= S_{par} \cdot S_{Pipeline} \\ &= \frac{T_{Seq}}{T_{Par}} \cdot Pipeline\ depth \\ &= \frac{P}{2 \cdot \lg P + 1} \cdot 2 \cdot \lg P \end{aligned} \tag{12}$$

Note that the $2 \cdot \lg P$ in the numerator approaches the $2 \cdot \lg P + 1$ in the denominator as P grows[5]. Thus, in the limit $S_{overall}$ approaches P, a linear speedup.

5 Implementation

In this section we outline pseudo-code for the core computation and communication pattern of the implementation of our parallel, pipelined reduction model. The while loop in the code below implements continuous, never-ending feature extraction as discussed in Section 4.5. The for loop and the parameter blksize control the communication pattern described and depicted in Section 4.3 (Figure 2). The if clause determines whether a processor sends or receives a message. It is these loops that are (software) pipelined and executed in parallel.

```
MPI_Init(&argc, &argv);
MPI_Comm_size(&size);
P=size;
MPI_Comm_rank(&rank);
output=NP_extractor(url);
while(true)
{
    blksize=2;
    for(i=1:i=lg(P):i++)
    {
      if(rank%blksize>0 and rank%blksize≤blksize/2)
      {
        buf=output;
        dest=rank+blksize/2;
        MPI_Send(buf,dest);
        output=NP_extractor(url);
      }
      else
      {
        source=rank-blksize/2;
        MPI_Recv(buf,source);
        list=buf;
        Merge(output,list);
      }
      blksize=blksize*2;
```

[4]Note that the constraint of equal pipeline stages is required for optimality of the pipeline operation as discussed previously in Section 4

[5]Note that no speedup is achieved from parallelism alone until $P \geq 8$, but as our results indicate speedups are achievable for $P < 8$ due to the presence of the pipeline

```
    }
}
```

6 Results

In this section we present empirical results that support the speedup model presented in Section 4.5. In the empirical results depicted below, the merge operation is insignificant and as a result we have a sequential execution time of

$$T_{Seq} = n \cdot T_{Comp} \tag{13}$$

The empirical results also do not include the final stage of the pipeline depicted in Figure 4, which is the merge or send to server. Thus we have

$$Pipeline\ depth = 2 \cdot \lg P \tag{14}$$

Due to the functional hazard outlined in Section 4.5, the pipeline depth is actually $2 \cdot \lg P - 1$.

Parallel execution time is now

$$T_{Par} = \frac{n}{\frac{P}{2}} \cdot T_{Comp} + \frac{n}{\frac{P}{2}} \cdot (T_{Merge} + T_{Comm}) \cdot (lgP - 1) \tag{15}$$

Therefore, the overall speedup is

$$S_{overall} = \frac{P}{4 \cdot \lg P - 2} \cdot (2 \cdot \lg P - 1) \tag{16}$$

Input Size	Runtime on 1 processor	Runtime on 4 processors	Speedup
50	120.67	87.64	1.37
150	258.74	160.95	1.60
450	652.51	357.66	1.82
1350	1816.80	940.87	1.93
4050	5310.16	2693.09	1.97
12150	15787.60	7925.34	1.99

Table 2. Speedup results on four processors (in seconds)

For 4 processors, $S_{overall} = \frac{4}{4 \cdot 2 - 2} \cdot (2 \cdot 2 - 1) = 2$.

For 8 processors, $S_{overall} = \frac{8}{4 \cdot 3 - 2} \cdot (2 \cdot 3 - 1) = 4$.

This speedup is confirmed by the results presented in Table 2 and Table 3. As the input size grows, the speedup approaches $S_{overall}$ as predicted by the speedup model.

Input Size	Runtime on 1 processor	Runtime on 8 processors	Speedup
50	120.67	71.02	1.69
150	258.74	105.32	2.45
450	652.51	210.90	3.09
1350	1816.80	505.84	3.59
4050	5310.16	1387.24	3.82
12150	15787.60	4009.91	3.93

Table 3. Speedup results on eight processors (in seconds)

7 Conclusion and Future Work

We have presented an analytical model for textual feature extraction in a highly distributed, massively parallel environment. The model can be employed to estimate parameters for optimally efficient use of a network of PCs/workstations. The model is thus suitable for use on a web-based mega-cluster such as that being developed by Data Synapse, Inc. [22] that scales to thousands of PCs in a highly distributed environment.

We have also presented a framework that combines the speedup achieved from both parallel and pipelined execution in one model. This is the only model that we are aware of that achieves a linear speedup with a parallelized associative operation (that involves a reduction). The class of applications that leverages this model is not "embarrassingly parallel" and as such encompasses a wide range of algorithms that heretofore have not been amenable to linear speedup.

The model is currently undergoing verification and scaling on a large number of PCs in a series of experiments being conducted by our research team at Lehigh University in conjunction with Data Synapse, Inc. We also extend a welcome to research efforts involved in the construction of extremely large clusters of PCs/workstations that have support for MPI-like runtime systems - we would like to perform additional scalability experiments on such platforms.

8 Acknowledgements

Co-author William M. Pottenger expresses his deep gratitude for his salvation to his Lord and Savior, Jesus Christ. We also gratefully acknowledge the assistance of Peter Lee and Jamie Bernardin, President and CTO of Data Synapse, Inc. In addition, we are quite grateful to Tianhao Wu and Faisal M. Khan for their diligent aid in coding and making the runs recorded in the results section of this article. Finally, our thanks to Denise Williams, HPDC-10 editor at IEEE, for her gracious extension of the deadline in the extenuating circumstances that existed at the time of writing.

References

[1] R. Bader, M. Callahan, D. Grim, J. Krause, N. Miller, and W. Pottenger. The role of the HDDI[tm] collection builder in hierarchical distributed dynamic indexing. In *Proceedings of Textmine '01 Workshop, First SIAM International Conference on Data Mining*, April 2001.

[2] F. Bouskila. The role of semantic locality in hierarchical distributed dynamic indexing and information retrieval. Master's thesis, University of Illinois at Urbana-Champaign, Department of Electrical and Computer Engineering, 1999.

[3] E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*. ACL, 1992.

[4] E. Brill. *A corpus-based approach to Language learning*. PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1993.

[5] E. Brill. Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

[6] C. T. Center. Fundamentals of distributed memory computing, 1999. www.tc.cornell.edu/Services/Edu.

[7] G. Cooke. Semantag. gcooke@rt66.com, http://www.rt66.com/gcooke/.

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R.Subramonian, and T. Eicken. LogP: A practical model of parallel computation. *Commun. ACM*, 39(11), 1996.

[9] D. Eager, J. Zahorjan, and E. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Trans. Computs.*, 38(3), 1989.

[10] H. Flatt and K. Kennedy. Performance of parallel processors. *Parallel Computing*, 12(1), 1989.

[11] I. T. Foster. *Designing and Building Parallel Programs*. Addison-Wesley, 1994.

[12] W. Francis and H. Kucera. Brown corpus manual. Department of Liguistics, Brown University, 1979 revision, http://www.hit.uib.no/icame/brown/bcm.html.

[13] A. Karp and H. Flatt. Measuring parallel processor performance. *Commun. ACM*, 33(5), 1990.

[14] F. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann, 1992.

[15] D. Nussbaum and A. Agarwal. Scalability of parallel machines. *Commun. ACM*, 34(3), 1991.

[16] D. Patterson and J. Hennessy. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, 1996.

[17] W. Pottenger. *Theory, Techniques, and Experiments in Solving Recurrences in Computer Programs*. PhD thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, May 1997.

[18] W. Pottenger, Y. Kim, and D. Meling. *Data Mining for Scientific and Engineering Applications*, chapter HDDI[TM]: Hierarchical Distributed Dynamic Indexing. Kluwer Academic Publishers, 2001.

[19] W. M. Pottenger. The role of associativity and commutativity in the detection and transformation of loop-level parallelism. In *Proceedings of the 12th ACM International Conference on Supercomputing*, July 1998.

[20] W. M. Pottenger and T. Yang. *Computational Information Retrieval*, chapter Detecting Emerging Concepts in Textual Data Mining. SIAM, 2001.

[21] J. Sanz and R. Cypher. Data reduction and fast routing: A strategy for efficient algorithms for message-passing parallel computers. *Algorithmica*, 1992.

[22] www.datasynapse.com.