Parallel Mining of Outliers in Large Database

Edward Hung*, David W. Cheung

Department of Computer Science and Information Systems,
The University of Hong Kong, Hong Kong.
email: ehung@cs.umd.edu, dcheung@csis.hku.hk

Abstract

Data mining is a new, important and fast growing database application. Outlier (exception) detection is one kind of data mining, which can be applied in a variety of areas like monitoring of credit card fraud and criminal activities in electronic commerce. With the ever-increasing size and attributes (dimensions) of database, previously proposed detection methods for two dimensions are no longer applicable. The time complexity of the Nested-Loop (NL) algorithm [13] is linear to the dimensionality but quadratic to the dataset size, inducing an unacceptable cost for large dataset.

A more efficient version (ENL) and its parallel version (PENL) are introduced. In theory, the improvement of performance in PENL is linear to the number of processors, as shown in a performance comparison between ENL and PENL using Bulk Synchronization Parallel (BSP) model. The great improvement is further verified by experiments on a parallel computer system IBM 9076 SP2. The results show that it is a very good choice to mine outliers in a cluster of workstations with a low-cost interconnected by a commodity communication network.

Keywords: Data Mining, Outlier Detection, Parallel Algorithm

1 Introduction

Data mining or knowledge discovery tasks can be classified into four general categories: (a) dependency detection (e.g. association rules [1]) (b) class identification (e.g. classification, data clustering [6, 14, 17]) (c) class description (e.g. concept generalization [7, 11]), and (d) exception/outlier detection [12, 13]. Most research has concentrated on the first three categories while most of the existing work on outliers detection has lied in the field of statistics [2, 8]. Although

*The author is currently in the Department of Computer Science at the University of Maryland, College Park, but the work in this paper was done when he was at the University of Hong Kong.

outliers have also been considered in some existing algorithms, they are not the main target and the algorithms only try to remove or tolerate them [6, 14, 17].

In fact, the identification of outliers can be applied in the areas of electronic commerce, credit card fraud detection, analysis of performance statistic of professional athletes [10] and even exploration of satellite or medical images [12]. For example, in a database of transactions containing sales information, most transactions would involve a small amount of money and items. Thus a typical fault detection can discover exceptions in the amount of money spent, type of items purchased, time and location. As a second example, satellite nowadays can be used to take images on the earth using visible lights as well as electromagnetic waves to detect targets such as potential oil fields or suspicious military bases. Detection of exceptional high energy or temperature or reflection of certain electromagnetic waves can be used to locate possible targets.

A simple algorithm called Nested-Loop algorithm (NL) has been proposed in [13], which, however, has a complexity of $O(kN^2)$, (k is the number of dimensions and N is the number of data objects), and the number of passes over the dataset is linear to N. By real implementation and performance studies, we find that the major cost is from the calculation of distances between objects. Though NL is a good choice when the dataset has high dimensionality, the large number of calculations makes it unfavorable. A cell-based algorithm has been proposed in [13]. It needs only at most three dataset passes. However, it is not suitable to high dimensions because its time complexity is exponential to the number of dimensions. NL always outperforms the cell-based algorithm when there are more than four dimensions [13]. In this paper, we will improve NL for high dimensional dataset, which is very common in data warehouse.

One approach to improve the NL algorithm is to parallelize it. In this paper, the definition of outliers and the original NL is described in the next few subsections in this introduction. NL is improved to reduce the number of calculations. The resulted algorithm, ENL, is given in Section 2. In Section 3, ENL is parallelized to further reduce the execution time in a shared-nothing system. In Section 4, the performance and improvement are analyzed by using Bulk Synchronization Parallel (BSP) model. After that, performance studies are given in Section 5. Finally, we give a discussion and related works in Section 6. This paper mainly focuses on identification of distance-based outliers, although our parallel algorithm can also be modified to perform the most expensive step of finding density-based outliers [4]. Related works on density-based outliers are described in Section 6.

1.1 Distance-Based Outliers

As given in [13], the definition of an *outlier* is as the following:

> *Given parameter p and D, an object O in a dataset T is a $DB(p, D)$-outlier if at least fraction p of the objects in T lies greater than distance D from O.*

From the definition, the maximum number of objects within distance D of an outlier O is $M = N(1 - p)$, where N is the number of objects. Let F be the underlying distance function that gives the distance between any pair of objects in T. Then, for an object O, the D-

neighbourhood of O contains the set of objects $Q \in T$ that are within distance D of O (i.e. $\{Q \in T \mid F(O,Q) \leq D\}$).

This notion of outliers is suitable for any situation in which the observed distribution does not fit any standard distribution. Readers are referred to [12] for the generalization of the notions of distance-based outliers supported by statistical tests for standard distributions. Works on density-based outliers are described in Section 6.

1.2 Assumption and Notation in NL Algorithm

Algorithm NL [13] is a *block*-oriented, nested-loop design. Here a *block* can involve one or more disk I/O, i.e. it may be necessary to take one or more disk I/O to read in a *block*. In this paper, a page needs one disk I/O access only. Thus, the total number of pages in a particular dataset is constant, but the total number of blocks in that dataset will change with the size of a block.

NL algorithm is designed for a uniprocessor system with one local memory and one local disk. The effect of cache is insignificant here. We make some assumptions here that are generally acceptable in real systems: there is no additional disk buffer from the operating system beside the buffer we will use in the algorithm; the disk access is sequential.

Let n be the number of blocks in the dataset T, k be the dimensionality, N be the number of objects in the dataset, P be the number of pages contained in a block, so the number of objects in a page is $N_P = \frac{N}{nP}$; let the time of accessing a page from the disk be $t_{\mathrm{I/O}}$, and the time of computing the distance between 2 objects be t_{comp}, which is linear to the dimensionality.

1.3 Original NL

The original NL algorithm from [13] with some clarification is described here.

Assume that the buffer size (for storing dataset) is $B\%$ of the dataset size. Then the buffer is first divided into 2 equal halves, called the first and second arrays. The dataset is read into the first or second array in a predefined order. For each object t in the first array, distance between t and all the other objects in the arrays are directly computed. A count of objects in its D-neighbourhood is maintained.

Algorithm NL

1. fill the first array (of size $\frac{B}{2}\%$ of dataset) with a block of objects from T
2. for each object t in the first array, do:
   (a) count the number of objects in first array which are close to t (distance $\leq D$); if the count $> M$, mark t as a non-outlier, where $M = N(1-p)$
3. repeat until all blocks are compared to first array, do:
   (a) fill the second array with another block (but save a block which has never served as the first array, for last)

(b) for each unmarked object t in the first array, do:

i. increase its count by the number of objects in second array close to t (distance $\leq D$); if the count $> M$, mark t as a non-outlier

4. report unmarked objects in the first array as outliers

5. if second array has served as the first array before, stop; otherwise, swap the names of first and second arrays and repeat the above from step 2.

The time complexity is stated as $O(kN^2)$ in [13], where k is the dimensionality and N is the number of objects in the dataset. The disk I/O time is considered briefly only in [13]. In fact, both the CPU time and I/O time should be considered. The detailed analysis of the computation time and disk I/O time is given below.

For computation time, the total time for calculation of distance between pairs of objects has an upper bound of $N^2 t_{\text{comp}}$, i.e. the number of calculations of distance is quadratic to the number of objects in the dataset. The actual number of calculations depends on the distribution of data, the location of data in the blocks, the distance D, and the number M, which in turn depends on the fraction p. Since in usual case, the number of outliers should be small, so M is small. No more calculation for a particular object t will be done if its count exceeds M. As a result, the actual number of calculations is much less than N^2, usually, within half, as shown in Section 5.

For disk I/O time, since the dataset is divided into $n = \left\lceil \frac{200}{B} \right\rceil$ blocks, the total number of block reads is $n+(n-2)(n-1)$, and the number of passes over the dataset is $\frac{n+(n-2)(n-1)}{n} = n-2+\frac{2}{n}$. The total number of page reads is $nP+(n-2)(n-1)P$. It is noted that P is directly proportional to the buffer size. With a fixed buffer size, n is directly proportional to N. So the I/O time $= (n+(n-2)(n-1))Pt_{\text{I/O}} > (n-2)^2 Pt_{\text{I/O}} = (\frac{N}{N_P P} - 2)^2 Pt_{\text{I/O}} \approx \frac{N^2 t_{\text{I/O}}}{PN_P^2}$, which has a complexity of $O(N^2)$.

Example 1

The following is an example.

Consider 50% buffering and 4 blocks of the dataset denoted as A, B, C, D, i.e. each block contains $\frac{1}{4}$ of the dataset. The order of filling the arrays and comparing is as the following:

1. A with A, then with B, C, D, for a total of 4 blocks reads;

2. D with D (no read required), then with A (no read), B, C, for a total of 2 blocks reads;

3. C with C, then with D, A, B, for a total of 2 blocks reads;

4. B with B, then with C, A, D, for a total of 2 blocks reads.

The table below shows the order of the blocks loaded, and the blocks staying in the two arrays of the buffer. Each row shows a snapshot after step 1 or step 3 (a).

	Block					Buffer	
disk I/O order	A	B	C	D		Array 1	Array 2
1	L					A	
2	*	L				A	B
3	*		L			A	C
4	*			L		A	D
5		L		*		D	B
6			L	*		D	C
7	L		*			C	A
8		L	*			C	B
9	L	*				B	A
10		*		L		B	D

L : Load

* : in buffer (unchanged)

The total number of disk I/O is $n + (n-2)(n-1) = 4 + (4-2)(4-1) = 10$ blocks. The total number of dataset passes is $\frac{10}{4} = 2.5$.

2 Enhanced NL

In NL, there are redundant block reading and comparison. In this section, a new order is proposed which results in reduced computation time and disk I/O time. The arrangement is that: in each turn, the blocks are read into the second array in a predefined order until the end of the series of ready blocks is reached, then the block in the first array is marked as done, the names of the two arrays are swapped and the order is reversed. The above is repeated until all blocks are done. The resultant is Enhanced NL (ENL) Algorithm, which is described below. For each object in the dataset, there is a count.

Algorithm ENL

1. **label all blocks as "ready" (the block is either in a "ready" or a "done" state)**
2. fill the first array (of size $\frac{B}{2}\%$ of dataset) with a block of objects from T
3. for each object t in the first array, do:
   (a) **increase the count by** the number of objects in first array which are close to t (distance $\leq D$); if the count $> M$, mark t as a non-outlier
4. **set the block-reading order as "forward"**
5. repeat until all **"ready"** blocks **(without marked as done)** are compared to first array **in the specified block-reading order**, do:
   (a) fill the second array with next block
   (b) for each object t_i in the first array, do:
      i. **for each object t_j in the second array, if object t_i or t_j is unmarked, then if dist$(t_i, t_j) \leq D$:**

A. **increase** count_i **and** count_j **by 1; if** $\text{count}_i > M$**, mark** t_i **as a non-outlier, proceed to next** t_i**; if** $\text{count}_j > M$**, mark** t_j **as a non-outlier**

6. report unmarked objects in the first array as outliers
7. if second array **is marked as done**, stop; otherwise, **mark the block in the first array as done, reverse the block-reading order**, swap the names of first and second arrays and repeat the above from step 3.

Although the time complexity of ENL is still $O(kN^2)$, where k is the dimensionality and N is the number of objects in the dataset, the cost of computation and disk I/O are both reduced compared with those of NL.

For computation time, the upper bound of the total time for calculation of distance between pairs of objects $= \frac{n(n+1)(N_P P)^2 t_{\text{comp}}}{2} = \frac{N(N+N_P P)t_{\text{comp}}}{2}$, which is still linear to the dimensionality. In original NL algorithm, only the count of the object in first array is updated. However, in ENL, the counts of both objects are updated for each comparison. Thus, the upper bound of the number of calculations of distances is reduced to almost half (compared with NL). For the actual reduction of the number of calculations, once again, it depends on the distribution of data, the locations of data in the blocks, the distance D, and the number M, which in turn depends on the fraction p. The comparions of performance of NL and ENL is shown in Section 5.

For simplicity, the upper bound of number of distance calculations between the objects in the same block was said to be $(N_P P)^2$ but in fact it only needs $\frac{(N_P P)(N_P P - 1)}{2}$ times.

For disk I/O time, if the dataset is divided into $n = \left\lceil \frac{200}{B} \right\rceil$ blocks, then the total number of block reads is $1+(n-1)+(n-2)+\cdots+2+1 = \frac{n(n-1)}{2}+1$, and the number of passes over the dataset is $\frac{\frac{n(n-1)}{2}+1}{n} = \frac{n-1}{2}+\frac{1}{n} \leq \frac{n}{2}$. The total number of page reads is $\left(\frac{n(n-1)}{2}+1\right)P$.

With a fixed buffer size, n is directly proportional to N. So the disk I/O time

$= \left(\frac{n(n-1)}{2}+1\right)Pt_{\text{I/O}} < \frac{n^2 P t_{\text{I/O}}}{2} = \frac{\left(\frac{N}{N_P P}\right)^2 P t_{\text{I/O}}}{2} = \frac{N^2 t_{\text{I/O}}}{2PN_P^2}$, which has a complexity of $O(N^2)$ and is nearly half of that of NL.

Example 2

Example 1 is extended here to illustrate ENL.

Consider 50% buffering and 4 blocks of the dataset denoted as A, B, C, D, i.e. each block contains $\frac{1}{4}$ of the dataset.

The order of filling the arrays and comparing is as the following:

1. A with A, then with B, C, D, for a total of 4 blocks reads;
2. D with D (no read required), then with C, B for a total of 2 blocks reads;

3. B with B, then with C, for a total of 1 blocks reads;

4. C with C, for a total of 0 blocks reads.

The table below shows the order of the blocks loaded, and the blocks staying in the two arrays of the buffer. Each row shows a snapshot after step 2 or step 5 (a).

	Block					Buffer	
disk I/O order	A	B	C	D		Array 1	Array 2
1	L					A	
2	*	L				A	B
3	*		L			A	C
4	*			L		A	D
5			L	*		D	C
6		L		*		D	B
7		*	L			B	C

L = Load

* = in buffer (unchanged)

The total number of disk I/O is $1 + 3 + 2 + 1 = 7$ blocks. The total number of dataset passes is $\frac{7}{4} = 1.75$.

3 Parallel ENL

PENL (Parallel ENL) is a parallel version of ENL, running in a shared-nothing system. Actually, when running in a processor, PENL is almost reduced to ENL. Most of block reading in ENL is replaced by transfer of blocks among processors through the communication network. Its major advantage is distributing the costly computations nearly evenly among all processors.

3.1 Assumptions and Notation

To extend the ENL to PENL, the assumptions and notations are extended. For simplicity, it is assumed that in the shared-nothing system, each node has only one processor. Each node has its own memory and local disk. The dataset is distributed equally in size to the local disk of each node without overlapping. Communication is done by message passing. The network architecture is designed such that each node can send message and receive message at the same time. (For simplicity of analysis in later section, we make the previous assumption. This requirement is not strict. At least it is required that internode communication is possible among nodes.) Besides, the nodes are arranged in a logical ring so that each node has two neighbour nodes. Logical arrangement means that physically the network can be in other architecture, e.g. bus, which does not affect the effectiveness of our algorithm, but only the performance.

Let n be the number of blocks in the dataset, n_P be the total number of pages of the dataset, k be the dimensionality, N be the number of objects in the dataset, P be the number of pages contained in a block, so $n_P = nP$, the number of objects in a page is $N_P = \frac{N}{nP}$, and the number

of objects in a block is $N_b = \frac{N}{n}$. Let the time of computing the distance between 2 objects be t_{comp}, the time of internode communication between 2 nodes to transfer a page of data be t_{comm}, and the time of accessing a page from the local disk be $t_{\text{I/O}}$. Let p be the number of processors or nodes, m be the size of local memory used for disk buffer.

To simplify the algorithm and analysis, it is assumed that the local memory buffers are all the same size for all nodes, and so are the data in local disk; and the number of data blocks in a local disk is an integer, i.e. number of pages in dataset (n_P) is a multiple of the product of the number of processors and the number of pages contained in a block (pP).

3.2 Algorithm

Each node has part of the dataset in its local disk. The number of pages of local data is $\frac{n_P}{p}$. The number of objects in local data is $\frac{N}{p}$. Each node has a local memory of size m, which is divided into 3 arrays. Each array can contain a block with the size of P pages. The first and second arrays function similarly to those in ENL, while the third array is used as a *temp buffer* to store data received from a neighboring node. Besides, a count of objects in D-neighbors for each object is maintained.

PENL is modified from ENL. The basic principle is: in a node, each time after a block is read from the node's local disk and the distance calculations are all done, then the block is transferred to the node's neighbor node and distance calculations are done using the block received from the node's another neighbour node. This is repeated until the block has passed all neighbors, then the node reads another block from the local disk and repeats the above again. Most of disk I/O operations are replaced by relatively fast internode communication. The huge number of calculations are now distributed by all nodes, which greatly reduces the execution time, i.e. the response time.

Algorithm **PENL**(node_id x)

1. label all blocks as "ready" (the block is either in a "ready" or a "done" state)
2. fill the first array with a block of objects
3. set the block-reading order as "forward"
4. set counter b to 0, set counter s to 0
5. repeat until all "ready" blocks are compared to first array in the specified block-reading order, do:
   (a) set counter c to 0
   (b) if $b = 0$, for each object t in the first array, do:
      i. increase the count by the number of objects in first array which are close to t (distance $\leq D$); if the count $> M$, mark t as a non-outlier
   (c) if $b = 0$, then set b to 1 and go to step (f)

(d) if $b \neq 0$, fill the second array with the next block

(e) for each object t_i in the first array, do:

i. for each object t_j in the second array, if object t_i or t_j is unmarked, then if dist(t_i, t_j) $\leq D$:

A. if $s = 0$ then increase count$_i$ by 1, otherwise, increase count$_i$ and count$_j$ by 1; if count$_i > M$, mark t_i as a non-outlier; if count$_j > M$, mark t_j as a non-outlier

(f) if $x = 0$ reverse the order of execution of steps (g) and (h)

(g) if $c = 0$, send the data in the first array to the neighbor node $(x + p - 1)$ mod p; otherwise, send the data in the second array to the neighbor node $(x + p - 1)$ mod p

(h) receive the data from the neighbor node $(x+1)$ mod p and store it in the temp buffer (third array)

(i) increment counter c by 1, swap the names of the second array and third array; if counter $c = p$, set $s = 1$ and continue the iteration in step 5, otherwise go to step 5(e)

6. if second array is marked as done, report unmarked objects in the first array as outliers; otherwise, mark the block in the first array as done, reverse the block-reading order, swap the names of first and second arrays and repeat the above from step 4.

The time analysis of PENL is given below briefly. Later a detailed analysis will be given using the Bulk Synchronous Parallel (BSP) model.

For each node, the operations are similar to ENL, except that each block is transferred to other nodes after computations are done on it. Thus, in brief, the upper bound of computation time in each node is $\left(\frac{\frac{n}{p}\left(\frac{n}{p}+1\right)}{2}\right) p(N_P P)^2 t_{\text{comp}} = \frac{N\left(\frac{N}{p}+N_P P\right) t_{\text{comp}}}{2}$.

Each node has $\frac{n}{p}$ blocks of local data. The local disk I/O time is the same as that of ENL executing with $\frac{n}{p}$ blocks of data, so the disk I/O time for each node is

$$\left(1+\left(\frac{n}{p}-1\right)+\left(\frac{n}{p}-2\right)+\cdots+1\right) Pt_{\text{I/O}} = \left(1+\frac{\frac{n}{p}\left(\frac{n}{p}-1\right)}{2}\right) Pt_{\text{I/O}}$$

$$< \left(\frac{\left(\frac{n}{p}\right)^2}{2}\right) Pt_{\text{I/O}} = \frac{\left(\frac{N}{N_P P}\right)^2 Pt_{\text{I/O}}}{2p^2} = \frac{N^2 t_{\text{I/O}}}{2p^2 P N_P^2}$$

For the internode communication time in a node, it is

$$p\left(\left(\frac{n}{p}\right)+\left(\frac{n}{p}-1\right)+\left(\frac{n}{p}-2\right)+\cdots+1\right) Pt_{\text{comm}}$$

$$= \frac{p\left(\frac{n}{p}+1\right)\left(\frac{n}{p}\right) Pt_{\text{comm}}}{2} = \frac{\left(\frac{n}{p}+1\right) nPt_{\text{comm}}}{2}$$

$$= \frac{\left(\frac{N}{pN_P P}+1\right)\left(\frac{N}{N_P P}\right) Pt_{\text{comm}}}{2} = \frac{\left(\frac{N}{pPN_P}+1\right) Nt_{\text{comm}}}{2N_P}$$

It is obvious that the upper bound of the computation time is linear to the reciprocal of the number of processors, the internode communication time decreases with increasing number of

processors, the disk I/O time is quadratic to the reciprocal of the number of processors. Please note that P changes with the size of buffer. If the total size of all local memory is fixed, i.e. pP is a constant, then the internode communication time in each node only varies with the dataset size, but not the number of processors, while the computation time and local disk I/O time in each node is linear to the reciprocal of the number of processors. Although only the upper bound of the computation time is shown, actually the calculations are distributed quite evenly among all the nodes, and so it is nearly linear to the reciprocal of the number of processors, as shown in the performance studies in Section 5.

3.3 Example

The following gives an example of execution of PENL on a dataset of 16 blocks using four nodes. Each node has four local blocks because the 16 blocks are evenly distributed in the four nodes. The four local blocks of node x are denoted as A_x, B_x, C_x, D_x.

The order of filling the arrays by disk I/O or internode communication and comparing in node 0 is as the following:

1. A_0 with A_0 (I/O), A_1, A_2, A_3 (3 communications), B_0 (I/O), B_1, B_2, B_3 (3 communications), C_0 (I/O), C_1, C_2, C_3 (3 communications), D_0 (I/O), D_1, D_2, D_3 (3 communications), for a total of 4 blocks reads and 12 communications;
2. D_0 with D_0 (no read required), D_1, D_2, D_3 (3 communications), C_0 (I/O), C_1, C_2, C_3 (3 communications), B_0 (I/O), B_1, B_2, B_3 (3 communications), for a total of 2 blocks reads and 9 communications;
3. B_0 with B_0 (no read required), B_1, B_2, B_3 (3 communications), C_0 (I/O), C_1, C_2, C_3 (3 communications) for a total of 1 blocks reads and 6 communications;
4. C_0 with C_0 (no read required), C_1, C_2, C_3 (3 communications) for a total of 0 blocks reads and 3 communications.

For each node, the number of disk I/O is $1 + 3 + 2 + 1 = 7$ blocks. The number of dataset passes is $\frac{7}{4} = 1.75$; the internode communication is $3(4 + 3 + 2 + 1) = 30$ times.

The table in Appendix A shows the details: the order of the blocks loaded in node 0 , the blocks transferred to and from node 0, and the blocks staying in the two arrays of the buffer of node 0.

If we run ENL using a single node with the same amount of memory as that in a node in PENL, then the total number of disk I/O will be $\frac{16(16-1))}{2} + 1 = 121$ blocks. The total number of dataset passes is $\frac{121}{16} = 7.5625$. The ratio of disk I/O in ENL to that in PENL using 4 nodes $= \frac{7.5625}{1.75} \approx 4.32$. The improvement is very significant. However, if we give the ENL the amount of memory same as the total sum of memory of all nodes, then we do not gain any benefit in the disk I/O because the size of a block is larger now, and so the total number of blocks will be much less than 16. Nevertheless, we still have a significant improvement in performance because the computation time, which is the major cost, is nearly evenly distributed by other nodes in PENL, as shown in Section 5.

3.4 Optimization

We can do the following optimization on PENL. For the outer iterations in step 5 (5(a) to 5(i)) with $b = 1$ and $s = 1$, we can only do approximately the first half of all inner iterations (5(e) to 5(i)) to reduce redundancy of block transmission and computation. For the example in Section 3.3, the inner iterations for the computations of blocks A_0 and A_3, D_0 and D_3, B_0 and B_3, C_0 and C_3 are skipped. Then the upper bound of the cost of computation of that outer iteration can be reduced from $p(N_P P)^2 t_{\text{comp}}$ to $\left(\left\lceil \frac{(p-1)}{2} \right\rceil + 1\right)(N_P P)^2 t_{\text{comp}}$. Therefore the upper bound of the total computation cost is reduced from $\left(\frac{n}{2p}\right)\left(\frac{n}{p}+1\right) p(N_P P)^2 t_{\text{comp}}$ to

$$\left(\frac{n}{2p}\right)\left(\frac{n}{p}+1\right) p(N_P P)^2 t_{\text{comp}} - \left\lfloor \frac{(p-1)}{2} \right\rfloor (N_P P)^2 t_{\text{comp}}$$
$$= \frac{N}{p} N_P P t_{\text{comp}} \left[\frac{1}{2}\left(\frac{n}{p}+1\right) p - \left\lfloor \frac{(p-1)}{2} \right\rfloor\right].$$

Therefore the computation-cost-reduction-ratio (the ratio of reduction of upper bound of computation cost by the optimization of PENL algorithm) $= \frac{\left\lfloor \frac{(p-1)}{2} \right\rfloor}{\frac{1}{2}\left(\frac{n}{p}+1\right) p} < \frac{\frac{(p-1)}{2}}{\frac{1}{2}\left(\frac{n}{p}+1\right) p} < \frac{1}{\frac{n}{p}+1}$ where $\frac{n}{p}$ is the number of blocks in each node. For the example in Section 3.3, $p = 4$, $n = 16$, so the reduction is 0.1. In practice, we have a large number of local blocks, so the reduction will not be significant.

In this paper, we will refer to the original PENL algorithm, unless specified, for the simplicity of implementation and analysis of the parallel algorithm

4 Theoretical Analysis using BSP Model

Before studying the performance of real execution of the algorithms, the theoretical analysis is given here. The BSP (Bulk Synchronous Parallel) model [3] is used to analyze the PENL algorithm because the hardware and software characteristics of the model match with PENL's platform requirement and working principle.

A BSP computer consists of a set of processor-memory pairs, a communication network that delivers messages in a point-to-point manner, and a mechanism for the efficient barrier synchronization of the processors. The BSP computer is a two-level memory model, i.e. each processor has its own physically local memory module and all other memory is non-local and accessible in a uniformly efficient way. PENL requires each node to have a local memory buffer. The accesses of other blocks in the buffers of other nodes are done by synchronous communication. Block transfers are done in a node-to-node manner.

The BSP computer operates in the following way. A computation consists of a sequence of parallel supersteps, where each superstep is a sequence of steps, followed by a barrier synchronization at which point any non-local memory accesses take effect. PENL requires a barrier synchronization for block transfers.

During a superstep, each processor has to carry out a set of programs or threads, and it can do the following: (i) perform a number of computation steps, from its set of threads, on values held locally at the start of the superstep; (ii) send and receive a number of messages corresponding to non-local read and write requests. During each superstep, PENL performs computations of items in one or two blocks, accesses disk for loading a new block, and executes block transmissions.

As a simple model that bridges hardware and software, BSP model provides portability across diverse platforms with predictable efficiency. It can be seen that the model is very suitable for PENL because PENL has coarse granularity and each superstep consists of a lot of distance calculations followed by message passing.

4.1 Cost Analysis

Define the following variables:

L: barrier, synchronization cost

d: ratio of the time (cost) of local disk I/O accessing an object $\left(\text{i.e. } \frac{\text{time of local disk I/O accessing a page}}{N_P}\right)$ to the time of computation on a distance between two objects

g: ratio of the time of internode communication of transferring an object $\left(\text{i.e. } \frac{\text{time of internode communication of transferring a page}}{N_P}\right)$ to the time of computation on a distance between two objects

Costs and barriers are added in the PENL algorithm, as shown below:

Algorithm PENL(node_id x)

1. label all blocks as "ready" (the block is either in a "ready" or a "done" state)
2. fill the first array with a block of objects

▷ Cost: dN_PP *(P pages in a block, N_P objects in a page)*

3. set the block-reading order as "forward"
4. set counter b to 0
5. repeat until all "ready" blocks are compared to first array in the specified block-reading order, do:
   (a) set counter c to 0, set counter s to 0
   (b) if $b = 0$, for each object t in the first array, do:
      i. increase the count by the number of objects in first array which are close to t (distance $\leq D$); if the count $> M$, mark t as a non-outlier

      ▷ Cost: N_PP

   ▷ Cost: $N_P^2P^2$

(c) if $b = 0$, then set b to 1 and go to step (f)

(d) if $b \neq 0$, fill the second array with the next block,

▷ Cost: $dN_P P$

(e) for each object t_i in the first array, do:

i. for each object t_j in the second array, if object t_i or t_j is unmarked, then if dist(t_i, t_j) $\leq D$:

A. if $s = 0$ then increase count$_i$ by 1, otherwise, increase count$_i$ and count$_j$ by 1; if count$_i > M$, mark t_i as a non-outlier; if count$_j > M$, mark t_j as a non-outlier

▷ Cost: $N_P^2 P^2$

(f) **set a barrier,** if $x = 0$ reverse the order of execution of steps (g) and (h)

▷ Cost: L

(g) if $c = 0$, send the data in the first array to the neighbor node $(x + p - 1)$ mod p; otherwise, send the data in the second array to the neighbor node $(x + p - 1)$ mod p

(h) receive the data from the neighbor node $(x + 1)$ mod p and store it in the temp buffer (third array)

▷ Cost: $gN_P P$

(i) **set a barrier,** increment counter c by 1, swap the names of the second array and third array; if counter $c = p$, set $s = 1$ and continue the iteration in step 5, otherwise go to step 5(e)

▷ Cost: L

6. if second array is marked as done, report unmarked objects in the first array as outliers; otherwise, mark the block in the first array as done, reverse the block-reading order, swap the names of first and second arrays and repeat the above from step 4.

Therefore the total costs of the algorithm is

$$\frac{N}{2}\left(\frac{N}{p} + N_P P\right) + \left(\frac{N}{2p}\left(\frac{N}{pN_P P} - 1\right) + N_P P\right) d + \frac{N}{2}\left(\frac{N}{pN_P P} + 1\right) g + \frac{N}{N_P P}\left(\frac{N}{pN_P P} + 1\right) L$$

The derivation can be found in Appendix B.

The first term is computation, the second one is disk I/O, the third one is communication, the last one is synchronization. Please notice that the time of computation is only a upper bound. Therefore this theoretical analysis does not give a reliable value of the actual execution time, but it still acts as a good reference for the comparison with ENL later.

When the block size, the page size and object size are constant, then it can be found that, if the dataset size is large ($N \gg pN_P P = \frac{n}{p}$, i.e. local block number is large),

- the computation cost is quadratic to the dataset size and linear to the reciprocal of the number of processors;
- the disk I/O cost is quadratic to the dataset size and quadratic to the reciprocal of the number of processors;

- the communication cost is quadratic to the dataset size and linear to the reciprocal of the number of processors;
- the synchronization cost is quadratic to the dataset size and linear to the reciprocal of the number of processors.

Please note that P changes with the size of buffer. If the total size of all local memories is fixed, i.e. pP is a constant, then all costs are still quadratic to the dataset size if the dataset size is large ($N \gg pN_PP = \frac{n}{p}$, i.e. local block number is large); besides,

- the computation cost is still linear to the reciprocal of the number of processors;
- the disk I/O cost is *now linear to the reciprocal of the number of processors*;
- the synchronization cost is *now linear to the number of processors.*

The above analysis tells us that when the total memory is fixed, then it is still beneficial to increase the number of processors as it is shown that the major cost, the computation, is linear to the reciprocal of the number of processors.

On the other hand, if the number of processors is kept unchanged, but the buffer size in each node (i.e. block size P) varies, then the computation cost is linear proportional to the number of pages in a block (P). Thus, with smaller block size, fewer computations are necessary. Besides, the local block number $\left(\frac{n}{p}\right)$ increases, which makes the computation-cost-reduction-ratio of the optimization of the algorithm (in Section 3.4) become smaller. However, it is not recommended to use a too small buffer because when P is too small, then $N \gg pN_PP$ and the effect of reduction of computation cost by small P will be very small. Besides, smaller block size also increases the cost of disk I/O, communication and synchronization.

4.2 Comparision of PENL with ENL

What about comparing PENL with ENL when both are given the same amount of memory?

For ENL, the corresponding cost is:

$$\frac{N(N+N_PP)t_{\text{comp}}}{2} + \left(\frac{n(n-1)}{2}+1\right)Pt_{\text{I/O}}$$

$$= \frac{N(N+N_PP_1)}{2} + \frac{\left(\left(\frac{N}{N_PP_1}\right)\left(\frac{N}{N_PP_1}-1\right)+2\right)N_PP_1d}{2}$$

P_1 is the number of pages in a block in ENL. In ENL, the buffer is divided into two arrays, so the total amount of memory is $2P_1$. In PENL, each local buffer is divided into three arrays, so the total amount of memory is $3pP$. Giving the same amount of memory to PENL and ENL, $2P_1 = 3pP$ i.e. $P = \frac{2P_1}{3p}$. Better implementation can be made so that it is sufficient to divide each local buffer into two arrays. However, here and in later sections, we will still choose three arrays in order to give advantage to the sequential algorithms for comparisons but we can still show that our parallel algorithm outperforms them.

Assume that, in a worse case, the ratio of number of calculations actually done in ENL to the total sum of number of calculations actually done in all nodes in PENL is $1:2$. Let f be the fraction of number of calculations actually done in ENL, so the fraction of that of PENL is $2f$.

In ENL, the computation cost is $C_{\text{ENL,comp}} = \dfrac{fN(N + N_P P_1)}{2}$

In PENL, the computation cost is $C_{\text{PENL,comp}} = fN\left(\dfrac{N}{p} + N_P P\right)$

$C_{\text{PENL,comp}} = fN\left(\dfrac{N}{p} + N_P\left(\dfrac{2P_1}{3p}\right)\right) = fN\left(\dfrac{N}{p} + \dfrac{2}{3}\left(\dfrac{N_P P_1}{p}\right)\right)$

When $p > 1$, $C_{\text{PENL,comp}} < \dfrac{fN}{p}(N + N_P P_1)$, so $C_{\text{PENL,comp}} < C_{\text{ENL,comp}}$ and $\dfrac{C_{\text{ENL,comp}}}{C_{\text{PENL,comp}}}$ is linear to the number of processors. Thus it is always a better choice to use PENL even if the total buffer size is fixed..

5 Performance Studies

5.1 Experimental Setup and Implementation

In our experiments, our base dataset is an 248-object dataset consisting of trade index numbers of HKSAR from 1992 to 1999 March [5]. Each object is from one of the four categories: imports, domestic exports, re-exports, and total exports. The four categories have equal number of objects. Each object has six attributes: index value, year-on-change percentage change of index value, index unit value, year-on-change percentage change of index unit value, index quantum, year-on-change percentage change of index quantum. Since this real-life dataset is quite small, and we want to test our algorithms on a large, disk-resident dataset, we generate a large number of objects simulating the distribution of the orginal dataset. In our testing, the distance D is defined so that the number of outliers is restricted to within a few percents of all objects to simulate the real situation.

The programs were run in an IBM 9076 SP2 system installed in the University of Hong Kong. The system consists of three frames. Each frame consists of 16 160 MHz IBM P2SC RISC processors. Individual node has its own local RAM (128 MB or 256 MB) and local disk storage (2 Gb, for system files and local scratch spaces). Each node in a frame is interconnected by high performance switches and the three frames are also linked up by an inter-frame high performance switch. The theroretical peak performance for each processor is 640 MFLOPS. In our tests, the sequential programs and parallel programs were run in dedicated mode using loadleveler (the batch job scheduler). In Section 5.2, NL and ENL were run in another system because the SP2 system has a time limit of 10 hours on running. It is a Sun Enterprise Ultra 450 with 4 UltraSPARC-II CPU running at 250MHz, with 1GB RAM and four 4.1GB hard disks.

In SP2, in order to make the comparison fair, we will fix the total amount of memory of all nodes for PENL to be the same as that for NL and ENL, which is able to hold 75000 objects. The number of objects is chosen so that the number of blocks in our test can be more reasonable.

As a result, the number of objects in a block ($N_P P$) for NL, PENL with 2, 4, 8, 16 processors is 37500, 12500, 6250, 3125, 1563 respectively. The number of objects is 50000, 100000, 200000, 400000, 800000, which implies that the number of blocks is 2, 4, 8, 16, 32 for PENL and 2, 3, 6, 11, 22 for NL.

We have implemented the three algorithms, NL, ENL and PENL using C. MPI (Message-Passing Interface) library was used in PENL for message passing among multiple processors [15].

For PENL, better implementation can be made so that it is sufficient to divide each local buffer into two arrays, rather than three arrays. However, we still chose three arrays for simplicity in order to give advantage to the sequential algorithms for comparisons but we can still show that our parallel algorithm PENL outperforms them.

It should be noted that for PENL, in each node, a part of memory is needed to act as counts for all objects. The decrease in memory to act as counts will decrease P (the number of pages in a block) and increase the total cost. However, the addition of the cost is small compared with the improvement from NL. It is because an object in a database usually contains tens or even hundreds of attributes, which may be integers, floating points or even strings. The size of a count is very small compared with the size of an object, so P will only decrease a bit. When the total number of objects are very huge, it is undesirable to hold all counts in the memory, then the counts can be stored in local disks. The total size of the counts is very small compared with the size of datasets. Thus, the extra disk I/O time accessing the counts affects the performance a bit only. In our implementation of ENL and PENL, we have the counts resident in disk and we load them only if they are required. This method is good, but it induces extra disk I/O In our experiments, we decide to define an object to have six dimensions (long integer data type), in order to make the effect of reading and writing the counts more significant. However, our results show that the effect is very minor, compared with the reduction of computation cost.

Besides, it needs extra communication to transfer the counts of objects to the node containing the objects in its disk, so that the outliers can be reported as soon as possible. The better way is that, in the end, the counts are gathered to a node and then the outliers are reported from combining the counts. The extra communication cost is little compared with the computation cost. We chose the second method in our implementation.

The final point to note is the computer architecture. Each processor has its own cache, so more the processors, the larger is the total cache capacity. Thus the hit ratio can be larger and the performance is further enhanced. Besides, with existing workstations, a cluster can be formed on them with low cost to perform PENL, rather than installing a new advanced but costly supercomputer. Although our experiments were conducted in a supercomputer, our results show that the commuication cost is very minor. Thus, communication network of low cost is sufficient.

5.2 NL vs ENL

In this section we will compare the performance of NL and ENL.

object number	2000		8000	
	NL	ENL	NL	ENL
execution time (s)	2.44	1.82	56.63	48.06
computation time (s)	2.41	1.82	56.56	48.01
disk I/O time (s)	0.00	0.00	0.02	0.02
calculation number	974467	618697	23095637	17763925
$\frac{\text{ENL execution time}}{\text{NL execution time}}$	0.7459		0.8487	

Figure 1: Table of comparisons of NL and ENL

object number	32000		128000	
	NL	ENL	NL	ENL
execution time (s)	978.70	845.16	15999.77	13989.09
computation time (s)	978.53	845.02	15999.01	13988.37
disk I/O time (s)	0.09	0.05	0.66	0.62
calculation number	396844218	314745232	> 2147483647	> 2147483647
$\frac{\text{ENL execution time}}{\text{NL execution time}}$	0.8636		0.8743	

Figure 2: Table of comparisons of NL and ENL (cont.)

From Figure 3 we can see that ENL is better than NL, although the improvement is not very great. Moreover, from the tables in Figure 1 and 2, it is found that the major cost is from the computation of distance, which can be greatly reduced in PENL as we will show later. Besides, we can see that the increase in execution time is approximately quadratic to the increase of objects, i.e. the execution time complexity is $O(N^2)$, indicating that it is very unlikely to use NL or ENL to deal with large number of objects. However, PENL can help to reduce the time.

5.3 Sequential vs Parallel

Here we will compare the performance of sequential program NL and parallel program PENL.

Figure 4 and 5 show that PENL with various number of processors outperforms NL, whatever the number of objects is. Even the number of processor is two only, the performance is improved by more than 100 percents. As we have said, the total amount of memory given to PENL is the same as that to NL, so it is very clear that PENL is always a better choice when a multiprocessor system or a cluster of workstations is available.

The result of NL with 400000 and 800000 objects and that of PENL with 2 processors and 800000 objects are not available because the execution time exceeds the time limit of a job in the SP2 system.

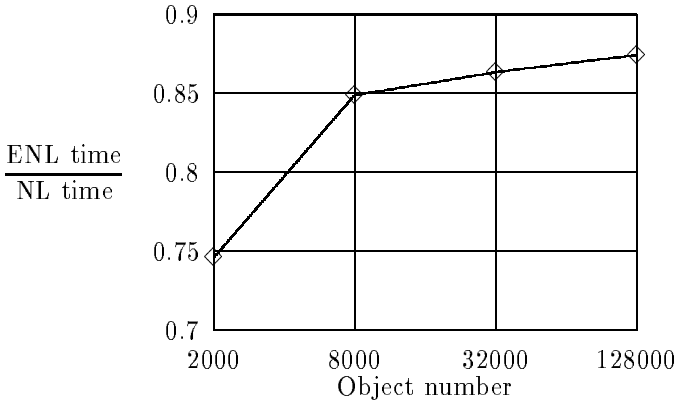

Figure 3: Comparison of execution time of NL and ENL

processor number	number of objects				
	50000	100000	200000	400000	800000
1(NL)	2445.30	2446.60	8419.80	N.A.	N.A.
2	302.95	1014.98	3990.31	15560.66	N.A.
4	177.92	517.62	2235.63	8588.52	33809.17
8	94.02	316.11	1148.35	4354.51	17058.63
16	48.53	158.21	574.09	2168.78	8391.13

Figure 4: Table of comparisons of execution time of NL and PENL in seconds

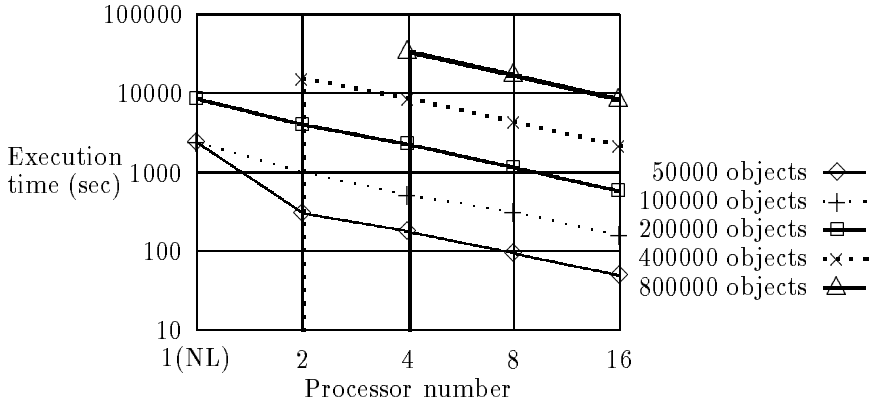

Figure 5: Execution time against number of processors

processor number	execution	CPU	I/O	communication	synchronization
1(NL)	2446.60	2446.48	0.0400	N.A.	N.A.
2	1014.98	1011.84	0.1112	0.5348	1.0148
4	517.62	515.42	0.1258	0.6978	1.2897
8	316.11	312.65	0.1277	0.9955	0.9110
16	158.21	152.03	0.1443	2.3178	1.2433

Figure 6: Table of comparisons of different costs in NL and PENL with 100000 objects in seconds

5.4 Variation of processor number

In this section we will see how the performance of PENL is related to the number of processors.

From Figure 4 and 5, we can see that the nearly straight lines are dropping steadily almost in parallel, indicating that the scalability is stable. In all cases, the execution time is almost halved when the number of processors is doubled, which is near to what our theoretical analysis predicts, i.e. the execution time is approximately linear to the reciprocal of the number of processors. Again, the increase in execution time is approximately quadratic to the number of objects, i.e. the execution time complexity is $O(N^2)$. But it should be noted that the execution time is linear to the dimensionality, thus it is still preferable for those database of high dimensionality.

5.5 Comparison of computation, disk I/O, communication time, synchronization time

In this section we will look more clearly into the contribution of execution time from the computation, disk I/O, communication and synchronization time in PENL.

Figure 6 shows that over 99 percents of the execution cost comes from computation time. Since PENL distributes the computation operations among all processors nearly evenly, so the execution time can be reduced greatly. Further improvement can be considered to focus on how to reduce the computation operations.

On the other hand, the disk I/O, communication and synchronization time are much more minor. Their trends are not the same as our theoretical analysis. The disk I/O time increases very slowly with the number of processors because there are more reading and writing of counts as the block size is smaller and the number of blocks loaded and transfered is larger (when a new block comes, the counts of the old block are written and the counts of the new block are read). The sum of the number of pages accessed by all processors should be close no matter how many processors are being used, but the total number of pages accessed for the counts increases with the number of processors as the counts are read and written more in times. Thus the disk I/O time increases a bit. The communication time and synchronization time depend much on the system at the moment of execution, e.g. the bandwidth and condition of the communication network.

6 Discussion and Related Works

NL algorithm is a straight forward method to mine outliers in a database. ENL proposed reduces both of the computation and disk I/O costs. Furthermore, the algorithm PENL is proposed to parallelize ENL. The analysis shows that if the total buffer size in the system is fixed, then the computation cost is linear to the reciprocal of the number of processors, which is verified by our performance studies. The great improvement is caused by the nearly even distribution of computation operations among all processors. Our performance studies further indicate that over 99 percents of the execution time comes from the computation, so the execution time is also linear to the reciprocal of the number of processors. These results show that PENL is very efficient comparing with NL and ENL, and further improvement can be focused on how to reduce the computation operations. Since other costs like communication time is very minor, so a low-cost cluster of workstations with commodity processors, interconnected by a low-cost communication network can be chosen as the platform of running PENL, rather than much more expensive supercomputer. A cluster is also much cheaper and easier to build, maintain and upgrade to achieve the similar performance that NL has in a single high performance processor system.

Breunig, et al. introduced a definition of a new kind of outliers (density-based outliers) and investigates its applicability [4]. Their heuristic can identify meaningful local outliers that the notion of distance-based outliers cannot find. The first step of computation of LOF (local outlier factor) is the materialization of the $MinPtsUB$-nearest neighborhoods (pages 102-103 of [4]).

Modification can be made in our parallel algorithm to perform that step, which is also the most expensive step in computation of LOF. Instead of updating the count of objects in D-neighborhood of each object, now each node stores the temporary $MinPtsUB$-nearest neighborhood of each object. The final $MinPtsUB$-nearest neighborhood of each object is obtained by combining all the temporary $MinPtsUB$-nearest neighborhoods of that object calculated by all nodes. We can choose to parallelize NL algorithm instead of using PENL algorithm for simplifying the implementation and reducing the disk storage space for temporary $MinPtsUB$-nearest neighborhoods. In that case, each node stores the $MinPtsUB$-nearest neighborhoods of objects in the block that stays in the first array only. The only difference is the reading order of blocks and the increase of number of blocks I/O and computation (by amost doubling).

Similarity search in high-dimensional vector space using the VA-File method outperforms other methods known [16]. Detection of outliers based on the VA-File is an approach different from the approaches of nested-loop or cell-structure. We will take that (using VA-File) into consideration in our future works.

References

[1] Agrawal, R., Imielinski, T., Swami, A. "Mining association rules between sets of items in large databases," *Proc. ACM SIGMOD*, pages 207-216, 1993.

[2] Barnett, V., Lewis, T. *Outliers in Statistical Data,* John Wiley, 3rd edition, 1994.

[3] Bisseling, R. H., McColl, W. F. "Scientific Computing on Bulk Synchronous Parallel Architetures," preprint 836, Dept. of Mathematics, Utrecht University, Dec. 1993, 31pp.

[4] Breunig, M. M., Kriegel, H. P., Ng, R. T., Sander, J. "LOF: Identifying Density-Based Local Outliers," *Proc. ACM SIGMOD 2000*, Dallas, TX, 2000.

[5] Census and Statistics Department, Hong Kong Special Administrative Region. "Trade Index Numbers", http://info.gov.hk/censtatd/hkstat/fas/ttrade2.htm, June 1, 1999.

[6] Ester, M., Kriegel, H. P., Sander, J., Xu, X. "A density-based algorithm for discovering clusters in large spatial databases with noise," *Proc. KDD*, pages 226-231, 1996.

[7] Han, J., Cai, Y., Cercone, N. "Knowledge discovery in databases: An attribute-oriented approach," *Proc. 18th VLDB*, pages 547-559, 1992.

[8] Hawkins, D. *Identification of Outliers,* Chapman and Hall, London, 1980.

[9] Hung, E., Cheung, D.W. "Parallel Algorithm for Mining Outliers in Large Database," *Proc. 9th International Database Conference (IDC'99)*, Hong Kong, July 15-17, 1999.

[10] Knorr, E. M. *On digital money and card technologies,* Technical Report 97-02, University of British Columbia, 1997.

[11] Knorr, E. M., Ng, R. T. "Finding aggregate proximity relationships and commonalities in spatial data mining," *IEEE Transactions on Knowledge and Data Engineering*, 8(6):884-897, 1996.

[12] Knorr, E. M., Ng, R. T. "A unified notion of outliers: Properties and computation," *Proc. KDD*, pages 219-222, 1997. An extended version of this paper appears as : Knorr, E. M., Ng, R. T. "A Unified Approach for Mining Outliers," *Proc. 7th CASCON*, pages 236-248, 1997.

[13] Knorr, E. M., Ng, R. T. "Algorithms for Mining Distance-Based Outliers in Large Datasets," *Proc. 24th VLDB*, 1998.

[14] Ng, R. T,, Han, J. "Efficient and effective clustering methods for spatial data mining," *Proc. 20th VLDB*, pages 144-155, 1994.

[15] Pacheco, P. S. "A User's Guide to MPI," Department of Mathematics, University of San Francisco, San Francisco, March 26, 1995.

[16] Weber, R., Schek, H. J., Blott, S. "A Quantitative Analysis and Performance Study for Similarity-Search Methods in Hifh-Dimensional Spaces," *Proc. 24th VLDB*, pages 194-205, 1998.

[17] Zhang, T., Ramakrishnan, R., Livny, M. "BIRCH: An efficient data clustering method for very large databases," *Proc. ACM SIGMOD*, pages 103-114, 1996.

APPENDIX

A Example of running of PENL

The table below shows the order of the blocks loaded in node 0 , the blocks transferred to and from node 0, and the blocks staying in the two arrays of the buffer of node 0. Each row shows a snapshot before step 5 (b) (if b=0) or step 5 (e).

	Block					Communication			Buffer		
disk I/O	A_0	B_0	C_0	D_0		to node 1	from node 3		Array 1	Array 2	Array 3
1	L								A_0		
						A_0	A_1		A_0	A_1	
						A_1	A_2		A_0	A_2	A_1
						A_2	A_3		A_0	A_3	A_2
2	*	L				A_3	A_0		A_0	B_0	A_3
						B_0	B_1		A_0	B_1	B_0
						B_1	B_2		A_0	B_2	B_1
						B_2	B_3		A_0	B_3	B_2
3	*		L			B_3	B_0		A_0	C_0	B_3
						C_0	C_1		A_0	C_1	C_0
						C_1	C_2		A_0	C_2	C_1
						C_2	C_3		A_0	C_3	C_2
4	*			L		C_3	C_0		A_0	D_0	C_3
						D_0	D_1		A_0	D_1	D_0
						D_1	D_2		A_0	D_2	D_1
						D_2	D_3		A_0	D_3	D_2
						D_3	D_0		D_0	A_0	D_3
						D_0	D_1		D_0	D_1	A_0
						D_1	D_2		D_0	D_2	D_1
						D_2	D_3		D_0	D_3	D_2
5			L	*		D_3	D_0		D_0	C_0	D_3
						C_0	C_1		D_0	C_1	C_0
						C_1	C_2		D_0	C_2	C_1
						C_2	C_3		D_0	C_3	C_2
6		L		*		C_3	C_0		D_0	B_0	C_3
						B_0	B_1		D_0	B_1	B_0
						B_1	B_2		D_0	B_2	B_1
						B_2	B_3		D_0	B_3	B_2
						B_3	B_0		B_0	D_0	B_3
						B_0	B_1		B_0	B_1	D_0
						B_1	B_2		B_0	B_2	B_1
						B_2	B_3		B_0	B_3	B_2
7		*	L			B_3	B_0		B_0	C_0	B_3
						C_0	C_1		B_0	C_1	C_0
						C_1	C_2		B_0	C_2	C_1
						C_2	C_3		B_0	C_3	C_2
						C_3	C_0		C_0	B_0	C_3
						C_0	C_1		C_0	C_1	D_0
						C_1	C_2		C_0	C_2	C_1
						C_2	C_3		C_0	C_3	C_2

L = Load

* = in buffer (unchanged)

B Derivation of total costs of PENL algorithm using BSP model

The total number of iterations of step 5(e) to 5(i) is

$$\left(\left(\frac{n}{p}-1\right)+\left(\frac{n}{p}-2\right)+\cdots+1\right)=\frac{\left(\frac{n}{p}\right)\left(\frac{n}{p}-1\right)}{2}.$$

Computation cost is $pN_P^2P^2\left(\left(\frac{n}{p}\right)+\left(\frac{n}{p}-1\right)+\cdots+1\right)=\frac{nN_P^2P^2}{2}\left(\frac{n}{p}+1\right)$

Disk I/O cost is $\left(1+\left(\frac{n}{p}-1\right)+\left(\frac{n}{p}-2\right)+\cdots+1\right)N_PPd=\left(\frac{n}{2p}\left(\frac{n}{p}-1\right)+1\right)N_PPd$

Communication cost is $p\left(\left(\frac{n}{p}\right)+\left(\frac{n}{p}-1\right)+\cdots+1\right)N_PPg=\frac{pN_PP}{2}\left(\frac{n}{p}+1\right)\left(\frac{n}{p}\right)g$

$=\frac{nN_PP}{2}\left(\frac{n}{p}+1\right)g$

Barriers are set before and after communication. The total number of synchronization (barrier) is $2p\left(\left(\frac{n}{p}\right)+\left(\frac{n}{p}-1\right)+\left(\frac{n}{p}-2\right)+\cdots+1\right)L=p\left(\frac{n}{p}+1\right)\left(\frac{n}{p}\right)L=n\left(\frac{n}{p}+1\right)L$

Therefore, by $n=\frac{N}{N_PP}$, the total costs of the algorithm is

$$\frac{nN_P^2P^2}{2}\left(\frac{n}{p}+1\right)+\left(\frac{n}{2p}\left(\frac{n}{p}-1\right)+1\right)N_PPd+\frac{nN_PP}{2}\left(\frac{n}{p}+1\right)g+n\left(\frac{n}{p}+1\right)L$$

$$=\frac{\left(\frac{N}{N_PP}\right)N_P^2P^2}{2}\left(\frac{\left(\frac{N}{N_PP}\right)}{p}+1\right)+\left(\left(\frac{\left(\frac{N}{N_PP}\right)}{2p}\right)\left(\frac{\left(\frac{N}{N_PP}\right)}{p}-1\right)+1\right)N_PPd$$

$$+\frac{\left(\frac{N}{N_PP}\right)N_PP}{2}\left(\frac{\left(\frac{N}{N_PP}\right)}{p}+1\right)g+\left(\frac{N}{N_PP}\right)\left(\frac{\left(\frac{N}{N_PP}\right)}{p}+1\right)$$

$$=\frac{N}{2}\left(\frac{N}{p}+N_PP\right)+\left(\frac{N}{2p}\left(\frac{N}{pN_PP}-1\right)+N_PP\right)d$$

$$+\frac{N}{2}\left(\frac{N}{pN_PP}+1\right)g+\frac{N}{N_PP}\left(\frac{N}{pN_PP}+1\right)L$$