ISSN: 0128-7680
Pertanika J. Sci. & Technol. **16 (2)**: 177 - 187 (2008)

Statistical Selection Algorithm for Peer-to-Peer System

Mohamed Othman and Kweh Yeah Lun*
Department of Communication Technology and Network,
Faculty of Computer Science and Information Technology,
Universiti Putra Malaysia, 43300 UPM, Serdang, Selangor, Malaysia
**E-mail: yeahlun@yahoo.com*

ABSTRACT

Over the years, the distributed database has been developed so fast that there's a need to develop an effective selection algorithm for it. Loo *et. al.* (2002) has proposed a statistical selection algorithm with the same objective and run in multicast / broadcast environment that has been proved that it is the best among others in terms of the number of messages needed to complete the searching process. However, this algorithm has a high probability of failure. A few improvements have been done to this original algorithm. This improved algorithm is developed based on the simulation of the real multicast environment. Modifications have been added in the improved algorithm to ensure that the unique pivot that never been used before is selected every time, and to solve problem that involve rank for certain key value that occur in more than one participant. Four performance measures have been conducted for the purpose of performance analysis between original and improved algorithm. These measures include probability of failure, number of messages needed, number of rounds needed and execution time. As a result, the probability of failure for the newly improved algorithm is 3.2% while the original algorithm is 19.2% without much overhead in increasing the number of messages and number of rounds needed.

Keywords: Distributed database, multicast, performance measure, selection algorithm

INTRODUCTION

The selection algorithm has been developed to ease distributed database operations and peer-to-peer computing. It can be applied to select the closest server. Selection operations are needed in some distributed sorting algorithms (Dechter, 1986; Huang, 1990; Wegner, 1984; Loo, 1991; Loo, 1995; Loo, 1997). In these distributed sorting algorithms, it is necessary to find the n/i^{th} keys (where $i = 1, 2, \ldots, p-1$; n is size and p is the number of computers involved). A file, F, with n records which is distributed in a few sites with all records totally ordered, and design resolution algorithms which minimize the amount of communication activity rather than the amount of processing activity, has been examined (Santoro, 1989). Different solutions and bounds exist for the distributed selection problem in the point-to-point network depending on the topology of the network (Frederickson, 1983; Matsuhita, 1983; Shirira, 1983). A general selection algorithm is developed using packets and the packet complexity is shown to be a significant improvement for a large range of packet sizes over the existing bounds (Negro, 1997). It is a bit optimal in star networks. The sampling techniques are used in designing the distributed algorithms (Rajasekaran, 1997). Some distributed algorithms (Saukas, 1988;

Received : 3 March 2008
Accepted : 26 June 2008
* Corresponding Author

Santoro, 1992; Huang, 1990) were suitable for the intranet environment. These algorithms consider the availability of the broadcast / multicast communication in the system. Wei (2002) has developed an efficient selection and sorting scheme for processing large files distributed over a network.

Many other distributed selection algorithms have been developed for various purposes based on different topologies and assumptions. Most researchers work on distributed selection algorithms that run on special topologies. Rodeh (1982) presented an algorithm for the case where two computers are connected together. Shen (1991) presented his algorithm on hypercube topology and Hao (1992) presented an algorithm on mesh topology. Aggrawal (1995) presented a selection algorithm for pyramid topology. However, the topologies of this algorithm are different in the Internet / intranet environment. They need to be modified before they are applied in the Internet / intranet environment. In 2003, Wu (2003) designed a fast and scalable parallel algorithm for selection and median filtering. It is a most time efficient algorithm, especially compared to the algorithms designed by Han (2002) and Pan (1994). Alexandros (2003) presented a randomized selection algorithm whose performance is analyzed in an architecture independent way on the bulk-synchronous parallel (BSP) model of computation along with an application of this algorithm to dynamic data structures, namely parallel priority queues. Bader (2005) presented an efficient randomized high-level parallel algorithm (Fast and UltraFast) for finding the median, given a set of elements distributed across a parallel machine, which solves the general selection problem that requires the determination of the element of rank k, for an arbitrarily given integer k.

Loo (2002) analyzed the problems in database operations and presented statistical selection algorithms which are designed to select the j^{th} smallest key from a very large file distributed over many computers. The algorithm aims to minimize the frequency of communication necessary to solve the selection problem. It is assumed that all data are sorted and within a certain range in each local file (Loo, 2002). All computers that participate are equal and they have equal share of the workload. The complexity analysis based on the number of messages needed is done to compare this algorithm with the previous algorithm (Huang, 1990; Santoro, 1992).

Loo's algorithms are efficient in terms of the number of communication messages where the number of communication messages required in the statistical selection algorithm for distributed databases is lower than in any published algorithms with the same type of computer connection topologies (Loo, 2002).

Loo presented an efficient distributed multiple selection algorithm designed to select multiple keys simultaneously from different data sets which are distributed to many computers in a peer-to-peer system and aimed at reducing the number of communication messages (Loo, 2005).

This research work is based on the work by Loo (2002) which is currently the best single selection algorithm in the peer-to-peer environment. The algorithm implemented in this research is a type of statistical selection algorithm derived from improvement of an existing algorithm presented by Loo (2002). Based on the simulation that has been set up, a few problems were found. The algorithm fails at a high rate where it cannot find the key, and instead the algorithm will cause the execution to keep on looping (infinite loop) to find the key. The improvements are done based on the analysis, results and problems found in the simulation of the original algorithm. These improvements aim to reduce the probability of failure of the original algorithm and at the same time maintain the number of messages needed in the whole selection process.

SEARCH PROCESS

The following search process is based on the equations from previous work (Loo, 2002) on which the algorithm is based with modifications . First, the coordinator will calculate the initial delimiter by gathering all the information regarding the smallest key and the largest of each participant's local file by using the following equation:

$$\text{Delimiter} = m + (M - m) * i / p \quad (1)$$

where $1 \leq i \leq p - 1$ and $i = 1$
M: maximum of the key values in the global file
m: minimum of the key values in the global file
p: number of participants involved

After calculating the initial delimiter value, the coordinator sends it to all other participants. Based on this delimiter value, each participant selects a *Pivot[i]* from the key that they generated in the local file, where this key is the biggest key less than or equal to the delimiter and i is and index reference to the participant number. The rank of a key is denoted as *R[i, r]* which is the number of keys that are smaller than the pivot i for participant r. Participant i will send its pivot to other participants, so concurrently there are $p - 1$ recipients for this pivot. Each participant will compare this pivot with all the keys stored in their local file to obtain the rank for it. These ranks will be broadcasted to the other participants sequentially. Each participant will receive p-1 ranks from the other participants. These ranks are summed up to obtain the global rank for its pivot. Global rank is defined by the following equation:

$$G = \sum_{r=1}^{p} R[i,r] \quad (2)$$

If this rank is equal to target rank, j, then the process will end. Otherwise another pivot needs to be generated. At this point, the local file is split into two sub-files based on this criterion. The first sub-file contains all keys that are smaller than its pivot and the second sub-file contains all the keys that are greater than the pivot. The first sub-file is used for the following operations if the global rank, G for the pivot is less than the target rank, j. If the global rank, G for the pivot is greater than the target rank, j, then the second sub-file is used. Note that, the current pivot is not included in both sub-files. A search value is calculated using the following equation:

*New value = old pivot value + (largest key of the sub-file – smallest key of the sub-file) * offset /*

(number of keys in the sub-file) (3)

$$\text{where} \quad \textit{offset} = \frac{(j - \sum_{r=1}^{p} R[i,j])}{p} \quad (4)$$

This new search value is used as a guideline to find the next pivot in the local file. So, if the first sub-file is used, the greatest key in the remaining sub-file, which is smaller or equal to the new search value is chosen as the new pivot, else if the second sub-file

is used, the smallest key in the remaining sub-file, which is greater than or equal to the new calculated value, is identified as the new pivot. The process is repeated until the target key is found.

When the intended target key is found, the participant holding that key will notify the coordinator. Then, the coordinator will stop the search process by breaking the iteration and display the result in an output window.

MODIFICATIONS MADE TO THE ALGORITHM

Three modifications have been made to the algorithm aimed at solving the problem stated in the statement of problem.

Modification 1 placed more restriction on the key values generated by the participants. No key values can be repeated in a local file. This means that the participants cannot generate two same random numbers and store it in the local file. However, the same key value can be repeated in the global file or other participants. Number '0' is treated as a key value like other values. It does not mean nothing or null value to the system.

Modification 2 aims to solve the problem caused by the repeated key that has the same value as the target key in the global file. The second modification made to the system is with regards to finding the rank for a pivot from another participant. As stated in the assumptions, the global ranks for the repeated keys in the global file are not the same. An additional assumption needs to be made here. Let's say, there are two participants namely Client1 and Client2 who have the same key value, k in their local file. If the global rank for k in Client1 is r, then the global rank for k in Client2 is $r + 1$. Generally, for pivot k from participant p, with its global rank being r and let's say there are a few participants $p - n$, where n is a natural number and $n < p$, carrying the same key value as the pivot k, then the global rank for the key value situated in the participant closest to participant p (which means that in $p - n$, for some value n, where $p - n$ is the greatest) is $r - 1$. Now, let's say that there are a few participants $p + n$, carrying the same key value as the pivot k, then the global rank for the key value situated in the participants closest to participant p (which means that in $p + n$, for some value n, where $p + n$ is the smallest) is $r + 1$. For example, let say 104 can be found in Client1, Client3 and Client4. The global rank for 104 in Client 3 is 29, then the global rank for 104 situated in Client1 is 29 – 1 = 28 and the global rank for 104 in Client4 is 29 + 1 = 30.
The algorithm for modification 2 in the client computers is as follows:

receive pivot
check the rank of the pivot
if receiver id > sender id
then set loop to run from the first key until
the last key in the local file
if received pivot >= current test key
then increment counter for rank
increment the loop counter
else end
else
set loop to run from the first key until the last key
in the local file
if received pivot > current test key

then increment counter for rank
increment the loop counter
else end

Modification 3 ensures that a unique pivot is generated each time. A virtual buffer fixed size is used to store the previous value that has been generated. Each time before a new pivot is identified, it is compared to the values in the buffer to make sure that it is unique. The algorithm for modification 3 is as follows:

Calculate new search value based on Eqs (3) and (4)
Select smallest or greatest key that is more than or less than the new search value based on the subfile that will be use as new pivot
Set loop to run from the first key until the last key in the buffer
if new pivot == buffer[counter]
then set same = true
increase index by one
pivot = key[index]
local rank = index + 1
break
increment counter
pivot selected and saved it inside the buffer
local rank = index + 1

THE SIMULATION ENVIRONMENT

At this moment, the improved algorithm developed can only be run in the simulation environment that tries to simulate the following situations where a single broadcast channel connects all participants, a large file is physically distributed among all the participants uniformly. The keys follow a known distribution; in this case they are in uniform distribution and sorted in ascending order. Java programming language is used to implement this simulation. All keys are generated randomly based on the built-in random number generator in Java. Each Java class file represents a peer in the system. All these files are running at the same time from the beginning of the process. There is one coordinator program that coordinates the flow of the whole process. Each Java class file that acts as a peer will generate its own random number and store it in its local file. Then the delimiter and pivots are calculated based on the algorithms stated above.

Both the original and improved algorithm implemented in the simulation only focuses on finding the 30^{th} smallest key in the global file that is being distributed among participants that vary from 2 to 6. The number of participants involved in the search process does not include the coordinator. Each participant holds 30 data that have been generated randomly and sorted at the beginning of the search process. The performance and complexity analysis for both algorithms is measured based on four criteria. They are probability of failure, number of messages needed, number of rounds needed and execution time.

PERFORMANCE MEASUREMENTS

Performance measure has been conducted to compare the performance between the improved algorithm and the original algorithm in various perspectives. These perspectives

are the probability of failure, number of messages needed, and number of rounds needed. Probability of failure studies the likeliness of the algorithm to fail to find the target key in a particular search that will affect the performance of the algorithm in real applications. Number of messages needed measures how many communication messages are needed between the peers in the whole selection process. This can be used as an indicator of the cost needed when this algorithm is run in the real environment. Number of rounds needed measures how fast the algorithm can find the key and it can be used to indicate the time needed for the selection process as more rounds needed indicate that more time is needed.

Probability of Failure

The original algorithm and the improved algorithm might fail in various ways due to the problems that may come from equations like equations (3) and (4) or other factors that have already been mentioned in the previous section. To calculate the probability of failure for both algorithms, each of the algorithms is executed 50 times. The result is shown in Table 1 and *Fig. 1*.

In *Fig. 1*, it is shown that the probability of failure of the improved algorithm is lower than 0.05 compared to the probability of failure of the original algorithm which ranges from 0.1 to 0.3. The average probability of failure of the improved algorithm is 0.032 while in the original algorithm it is 0.192.

It is assumed that failure has occurred when the algorithm cannot find the target key within 20 rounds. The main factor contributing to this is the failure of equation (3). At certain points of execution, the equation may obtain the result for the new pivot that appears to be the previous pivot that has been used, then the next pivot calculated will be exactly the same as the previous pivot too, thus looping occurs and causes failure.

TABLE 1
Probability of failure

Algorithm	Number of participants				
	2	3	4	5	6
Original	0.26	0.18	0.12	0.18	0.22
Improved	0.02	0.04	0.04	0.02	0.04

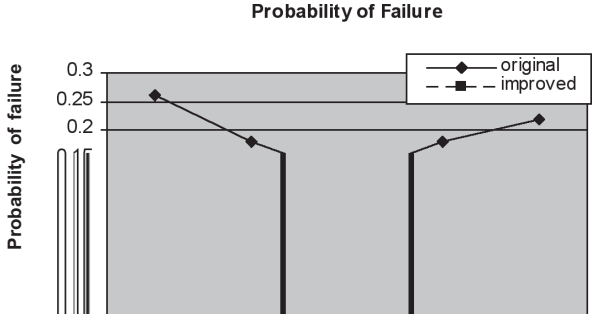

Fig. 1: Probability of failure

For the improved algorithm, the probability of failure is very low and consistent compared to the original algorithm. This improved algorithm is able to avoid the search process from looping by choose another new pivot if the current generated pivot is one that has been used before.

The Number of Messages Needed

The expected number of messages needed is also calculated based on a previous work by Loo (2002). The results are as shown in Table 2 and *Fig. 2.*

TABLE 2
Number of messages needed

Algorithm	Number of participants				
	2	3	4	5	6
Original	10.34	14.64	18.9	22.5	28.46
Improved	12.92	18.76	23.64	30.84	33.58

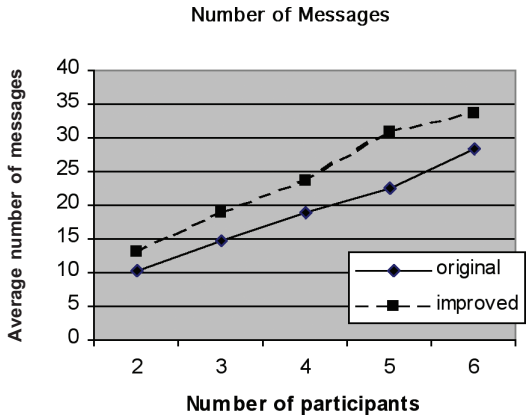

Fig. 2: Number of messages needed

For both algorithms, the number of messages increased as the number of participants increased in the system as shown in *Fig. 2.* This is because when more participants are involved, more messages are passed between them in each round and therefore more data are generated and presented in the global file. So, more pivots needed to be tested before the target key can be found. This increases the number of messages needed.

Depending on the distribution of the data to each participants, it may take more rounds to find a certain key or less than expected and thus, the number of messages needed to find the 30^{th} key is sometimes more or less, but the result is more than expectated most of the time as shown in the graph. However, as shown in the graph, the number of messages needed in the improved algorithm is more than that required in the original version by approximately 2 to 8 messages. This is because the improved algorithm can be run in most cases including cases where the data distribution is quite

complicated. For cases where the data distribution is complicated more messages are needed to find the target key.

Number of Rounds Needed

For both algorithms, the number of rounds needed for completion decreases as the number of participants increases due to the following factors. As stated in the previous section, one round is defined as the time the first participants finishes its turn, followed by the next participant and so on until the last participant. When more pivots are checked in one round, the number of rounds needed to find the target key will be less. The result is shown in Table 3 and *Fig. 3*.

The improved algorithm always utilises more number of rounds taken than the original algorithm. This is because the improved algorithm is successful in almost all types of data distributions. Different types of data distribution require different number of rounds to find the target key, so the average number of rounds taken by the improved algorithm takes into consideration all the different types of data distribution, from simple to complicated ones.

TABLE 3
Number of rounds needed

Algorithm	Number of computers				
	2	3	4	5	6
Original	3.34	3.12	3.04	2.98	2.88
Improved	3.92	3.72	3.46	3.34	3.18

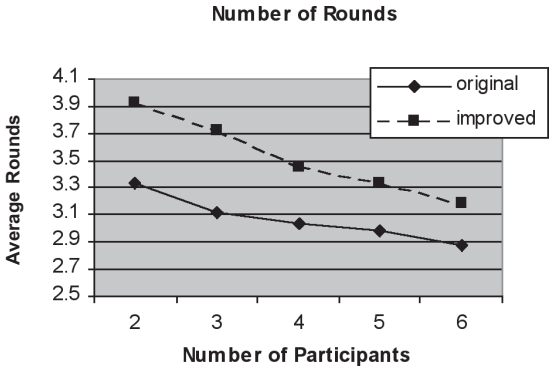

Fig. 3: Number of rounds needed

DISCUSSION ABOUT MODIFICATIONS

In modification 1, no repeated key is allowed in the same local file but it is allowed in the global file. This is because in a distributed system environment especially the distributed database system, the repeated keys are quite common in the whole system where each database system at different sites may have the same keys (data) for the

purpose of data redundancy and reliability. However, there is no point in having the repeated keys in the same local file (database) as the keys in the database should each be unique.

In modification 2, the global ranks for repeated keys in different peers must be unique so that the selection process can be made simple without much execution time wasted in determining which peer is carrying the repeated key to be selected if all repeated keys have the same global rank. Modification 2 aimed to solve this problem by placing each peer in sequence where the global rank of the repeated key in the local file of the peer in lower sequence is smaller than the global rank of the repeated key in the local file of the peer in higher sequence.

In modification 3, when the file is broken up into two sub-files after finding that the pivot is not the target key, the keys in the second sub-file will include the pivot itself while the first sub-file does not include this pivot. In the modification, both sub-files will not contain the current pivot. This is to avoid the problem of selecting the same pivot as the next pivot. As stated in the methodology section, for both the original and improved algorithms, the criteria to find the next pivot depends on which sub-file is going to be used. If the first sub-file is used, then the greatest pivot that is smaller or equal to the search value is chosen, else if the second sub-file is used, then the smallest pivot that is greater or equal to the search value is chosen. This is in contrast with the original methodology provided by Loo (2002), which stated that the smallest key that is greater or equal to the search value is selected as the new pivot for both sub-files. The reason that these changes are made is to ensure that the newly selected pivot comes from the sub-file that is going to be used. The original methodology will lead to a situation where the next pivot selected is the same as the previous pivot.

To eliminate the looping problem, a buffer of size 10 is used. This buffer records the pivot that has already been used. After the new pivot has been selected from the sub-file, it is compared to the values stored in the buffer, if it is the same as one of the previous pivots, then a new value is selected by adding one to the index of the current pivot, and thus the next pivot selected is one that is greater and next in sequence to the current pivot. This will ensure that the pivots selected will always be unique and different.

However, this modification does have some limitations to the system. The modification that has been made in the improved algorithm takes up more space in the memory and requires longer execution time. If the search process takes up more than 10 rounds, then the pivot in the first index of the buffer is overwritten by the next pivot that is going to be recorded in 11^{th} round. Thus, the looping problem still cannot be totally avoided. In this case, the pivot recorded in the first index will be lost and the pivot selected in the future may be equal to the pivot that has been erased from the buffer.

CONCLUSIONS AND FUTURE STUDIES

The improved algorithm has decreased the probability of failure present in the original algorithm. However, some of the added new features in the improved algorithm consume more messages than the original algorithm and some of the cases require more rounds to be complete to find the target key. Although the number of messages and the number of rounds needed has been increased in the improved algorithm, the execution time for both algorithms is almost the same and falls within a certain range for all cases with a different number of participants in the system.

Future studies include further modification of the algorithm to reduce the probability of failure to zero, and select other k^{th} smallest key to compare their performance.

REFERENCES

AGGRAWAL, C., JAIN, N. and GUPTA, P. (1995). An efficient selection algorithm on the pyramid. *Journal of Information Processing Letters, 53,* 37-47.

ALEXANDROS, V., GERBESSIOTIS, CONSTANTINOS and SINIOLAKIS, J. (2003). Architecture independent parallel selection with applications to parallel priority queues, *Theoretical Computer Science, 301,* 119 – 142.

BADER, D. (2004). An improved, randomized algorithm for parallel selection with an experimental study. *Journal of Parallel and Distributed Computing, 64,* 1051-1059.

DECHTER, R. and KLEINROCK, L. (1986). Broad communications and distributed algorithms, *Journal of IEEE Transactions on Computers, C-35*(3), 123-128.

FREDERICKSON, G.N. (1983). Tradeoffs for selection in distributed networks. In *Proceedings of 2nd ACM Symposium Principles Distributed Computing* (pp. 154-160). Montreal, Quebec, Canada.

HAN, Y., PAN, Y. and SHEN, H. (2002). Sublogarithmic deterministic selection on arrays with a reconfigurable optical bus. *IEEE Transaction on Computers, 51*(6), 702 – 707.

HAO, M., MACKENZIE, P. and STOUT, Q. (1992). Selection on the reconfigurable mesh. In *Proceedings of 4th Symposium on the Frontiers of Massively Parallel Computation* (pp. 38-45). McLean, Virginia, USA.

HUANG, J.H. and KLEINROCK, L. (1990). Distributed selectsort sorting algorithms on broadcast communication networks, *Journal of Parallel Computing, 16*(2/3), 183-190.

LOO, A. (2005). Distributed multiple selection algorithm for peer-to-peer systems. *The Journal of Systems and Software, 78,* 234 – 248.

LOO, A., BLOOR, C. and GREY, D. (1997). Complexity analysis of distributed database algorithm. *Proceedings of High Performance Computing in Engineering 97,* Gran Canaria, Spain.

LOO, A. and CHOI, Y.K. (2002). A peer-to-peer distributed selection algorithm for the internet. *Journal of Internet Research: Electronic Networking Applications and Policy, 12*(1), 16-30.

LOO, A., CHUNG, C., FU, R. and LO, J. (1995). Efficiency measurement of distributed statistical sorting algorithms. *Proceeding of Applications of High Performance Computing in Engineering,* Milan, Italy.

LOO, A. and NG, J. (1991). Distributed statistical sorting algorithm. *Proceeding of IEEE Singapore International Conference on Networks(SICON)* (pp. 222-225). Singapore.

MATSUHITA, T.A. (1983). Distributed algorithms for election (Master Thesis, Department of Computer Science, University of Illinois, Urbana, Illinois, USA, 1983).

NEGRO, A and SANTORO, N. (1997). Efficient distributed selection with bounded messages. *Journal of IEEE Transactions on Parallel and Distributed Systems, 8*(4), 397-401.

PAN, Y. (1994). Order statistics on optically interconnected multiprocessor systems. *Proceeding of First International Workshop Massively Parallel Processing Using Optical Interconnections* (pp. 162 – 169).

RAJASEKARAN, S. and WEI, D.S.L. (1997). Designing efficient distributed algorithms using sampling techniques. *Proceedings of 11th International Parallel Processing Symposium (IPPS'97)* (pp. 397-401). Geneva, Switzerland.

RODEH, M. (1982). Finding the median distributively. *Journal of Computer and System Science, 24*(2), 162-167.

SANTORO, N., SIDNEY, J.B. and SIDNEY, S.J. (1992). A distributed selection algorithm and its expected communication complexity. *Journal of Theoretical Computer Science, 100,* 185-204.

SANTORO, N. and SUEN, E. (1989). Reduction techniques for selection in distributed files. *Journal of IEEE Transactions on Computers, 38*(6), 891-896.

SAUKAS, E. and SONG, S. (1988). Efficient selection algorithms on distributed memory computers. *Proceedings of High Performance Networking and Computing*, Orlando, Florida, USA.

SHEN, H. (1991). A universal algorithm for parallel k-selection in hypercubes. *Journal of Parallel Computing, 18,* 139-145.

SHIRIRA, L., FRANCEZ, N. and RODEH, M. (1983). Distributed k-selection: from a sequential to a distributed algorithm. *Proceedings of 2nd ACM Symposium Principles of Distributed Computing* (pp. 143-153). Montreal, Quebec, Canada.

WEGNER, L.M. (1984). Sorting a distributed file in a network. *Journal of Computer Networks, 8,* 451-461.

WEI, D.S.L., RAJASEKARAN, S., CHENG, Z. and NAIK, K. (2002). Efficient selection and sorting schemes using coteries for processing large distributed files. *Journal of Parallel and Distributed Computing, 62,* 1295 – 1313.

WU, C.H. and HORNG, S.J. (2003). Fast and scalable selection algorithms with applications to median filtering. *IEEE Transaction on Parallel and Distributed Systems, 14*(10), 983 – 992.