Published in final edited form as:

Proc IEEE Symp Comput Commun. 2013 July ; 2013: 000612–000617. doi:10.1109/ISCC.2013.6755015.

A Two-Pass Exact Algorithm for Selection on Parallel Disk Systems

Tian Mi and Sanguthevar Rajasekaran

Department of Computer Science and Engineering, University of Connecticut, Storrs, USA

Tian Mi: tian.mi@engr.uconn.edu; Sanguthevar Rajasekaran: rajasek@engr.uconn.edu

Abstract

Numerous OLAP queries process selection operations of “top N”, median, “top 5%”, in data warehousing applications. Selection is a well-studied problem that has numerous applications in the management of data and databases since, typically, any complex data query can be reduced to a series of basic operations such as sorting and selection. The parallel selection has also become an important fundamental operation, especially after parallel databases were introduced. In this paper, we present a deterministic algorithm *Recursive Sampling Selection (RSS)* to solve the exact out-of-core selection problem, which we show needs no more than $(2 + \varepsilon)$ passes (ε being a very small fraction). We have compared our *RSS* algorithm with two other algorithms in the literature, namely, the *Deterministic Sampling Selection* and *QuickSelect* on the Parallel Disks Systems. Our analysis shows that *DSS* is a $(2 + \varepsilon)$ -pass algorithm when the total number of input elements N is a polynomial in the memory size M (i.e., $N = M^c$ for some constant c). While, our proposed algorithm *RSS* runs in $(2 + \varepsilon)$ passes without any assumptions. Experimental results indicate that both *RSS* and *DSS* outperform *QuickSelect* on the Parallel Disks Systems. Especially, the proposed algorithm *RSS* is more scalable and robust to handle big data when the input size is far greater than the core memory size, including the case of $N \gg M^c$.

Keywords

Selection; Parallel Disk System; Median; OLAP queries

I. Introduction

Numerous OLAP queries process selection operations of “top N”, median, “top 5%”, in data warehousing applications [1]. The selection problem is to identify the i^{th} smallest element from out of N given input elements. In particular, finding the median has various applications in data analysis. Compared with the mean, the median is less sensitive to outliers. Some applications include noise reduction in signal or image processing (known as median filter [2]), high-breakdown regression in robust statistics [3], [4], clustering, neural networks, data mining, optimization problems, etc.

Hoare’s *QuickSelect* algorithm runs in an expected $\Theta(N)$ time, but it has a worst-case run time of $O(N^2)$. Blum, et al. proposed a deterministic algorithm, known as *median of medians* algorithm, which takes $O(N)$ time [5] in the worst case. The number of comparisons made

by Floyd and Rivest's randomized algorithm is $[N + \min(i, N - i) + o(N)]$, with high probability [6].

In parallel databases, the parallel selection has become an important fundamental operation, and has been well studied on different parallel models. Examples of randomized parallel selection algorithms are: 1) Constant time algorithms on the comparison tree model [7], [8] that are based on the ideas from [6]; 2) Constant time maximum selection algorithms on the CRCW PRAM [9]; 3) Algorithms for the star graph [10]; 4) Algorithms for the hypercube [11]; and 5) Algorithms on the mesh [12], [13]. [14], [15] give a survey on randomized selection algorithms.

Examples of deterministic algorithms include the

$O(\min\{p \log(N/p), \max\{N/p^{2/3}, \sqrt{p}\}\})$ -time algorithm on a p -node mesh by Krizanc and Narayanan [16], and the $O((N/p) \log \log p + (T_p^s + T_p^b \log p) \log(N/p))$ -time algorithm of Plaxton on a p -node hypercube (where T_p^s is the time for sorting p elements and T_p^b is the time for broadcasting in a p -node hypercube) [17]. Rajasekaran, et al., have presented a deterministic algorithm with a run time of $O(N/p \log \log p + T_p^s \log N)$ on any p -node network, and a randomized algorithm with an expected run time of $O((N/p + T_p^{sparse}) \log \log p)$ [18], where T_p^{sparse} is the time needed for collecting and sorting $p^{1-\varepsilon}$ sample keys using p processors and ε being a constant < 1 . Dynamic bucketing and randomized bucketing were used to find medians or quantiles in [19].

For numerous applications in science and engineering, the size of data increases much faster than the available core memory. As a result, out-of-core algorithms are becoming more and more important to handle big data. "Out-of-core" refers to the case when the size of input data is much greater than what the core memory can hold. I/O operations, which are much slower than in-core computing, dominate the running time of any out-of-core algorithm. Therefore, it is customary to only concentrate on minimizing the number of I/O operations while developing out-of-core algorithms.

Several approximate algorithms have been studied for out-of-core selection. For example, a one-pass algorithm to compute approximate quantiles has been given by Munro and Paterson [20]. Utilizing the knowledge on the data distribution, Alsabti, et al., have presented an algorithm to estimate the quantile values with provable error bounds [21]. A distribution independent quantile estimation [22] can also be done in one pass with error guarantees that costs less memory than [20], [21].

To alleviate the I/O bottleneck, the Parallel Disks System (PDS) model has been proposed by Vitter and Shriver [23]. In this model D independent disks can simultaneously operate. In particular, in one parallel I/O operation, a block of data of size B can be fetched into the main memory of a computer from each of the D disks. If M is the core memory size of the computer, it is usually assumed that $M \geq 2DB$. Numerous algorithms have been proposed for solving fundamental computational problems such as sorting, selection, graph search, etc. on the PDS model. In this paper, we are interested in developing efficient *exact* selection algorithms for the PDS.

In [24], Rajasekaran has presented two exact algorithms for selection on the PDS that make only a constant number of passes through the data. A ‘pass’ refers to the task of bringing (from the disks) into the core memory each input key exactly once. Note that in one parallel I/O operation, we can bring in DB keys into the main memory. If the data size is N , clearly, a

single pass corresponds to $\frac{N}{DB}$ parallel I/O operations. The randomized algorithm of [24] needs $O(N/(DB))$ I/O operations with high probability, and the deterministic algorithm costs $O(N/(DB))$ I/O operations, under the assumption that N is a polynomial in M . These algorithms also make another assumption as will be pointed out later. Our analysis indicates that under these two assumptions, the algorithms of [24] run in two passes through the data.

In this paper, we present a two-pass *exact deterministic* selection algorithm on the PDS. We also provide theoretical analysis on the proposed algorithm, the deterministic sampling algorithm in [24], and the classic *QuickSelect* algorithm on the PDS. We have experimentally evaluated these three algorithms, and these results show that the proposed algorithm outperforms the others.

II. Definitions and Notations

In this section we introduce some notations and definitions.

Let N be the total number of input elements, M be the total number of elements the core memory can hold, D be the number of disks in the PDS, and B be the block size. The i^{th} smallest element to be identified is called the *target* and is denoted by t_i .

The algorithm of [24] employs deterministic sampling (as shown in Figure 1). The sampling is done in levels. At level 0, we have all the input elements. We partition the elements at this level into groups of size M each, sort each of these groups, and send every $\sqrt{M}^{th}$ element of each group to level 1. The keys that move to level 1 constitute a deterministic sample of

keys at level 0. If there are N keys at level 0, then, $\frac{N}{\sqrt{M}}$ keys move to level 1. Then the keys in level 1 are partitioned into groups of size M each in the same manner, each group is sorted, and every $\sqrt{M}^{th}$ key of each group moves to level 2 as a part of a deterministic

sample of keys at level 1 and therefore the number of keys in level 2 is $\frac{N}{M}$. This process of sampling continues to as many levels as it takes to have M keys or less. From this sample, we pick two keys u and v such that the i^{th} smallest key of the input has a value in the range $[u, v]$. Having found u and v , we delete all the input keys that do not have a value in the range $[u, v]$. This process of successive sampling and deletion is referred to as an *iteration*. On the keys that survive the first iteration we apply a similar series of samplings, identify two keys u' and v' that bracket the key to be selected, and delete the keys that do not have a value in the range $[u', v']$. This completes the second iteration. We perform as many iterations as needed to bring down the number of surviving keys to M . At this point we bring all the surviving keys into the core memory, perform an in-core selection and output the right element.

Definition 1

In the deterministic sampling, the set of keys in level j is denoted as L_j . The set of keys in the top level of the tree is denoted as L_m . Note that $L_m = M$.

Definition 2

A run is a sorted group of elements (of size M , typically). The k^{th} run in L_i is denoted as $R_i^k = \{r_1, r_2, \dots, r_M\}$ where $|R_i^k| = M$.

Definition 3

A $\sqrt{M}$ -sample of a run $R = \{r_1, r_2, \dots\}$ is a set of every $\sqrt{M}^{th}$ element of R , i.e., $\{r_{\sqrt{M}}, r_{2\sqrt{M}}, r_{3\sqrt{M}}, \dots\}$.

Definition 4

We denote the number of I/O operations taken by any selection algorithm on N elements as $IO(N)$. The number of read I/O operations and the number of write I/O operations are denoted as $IO^r(N)$ and $IO^w(N)$, respectively. The number of passes needed for selection from N elements is denoted by $P(N)$. Also, the number of read passes and the number of write passes are denoted as $P^r(N)$ and $P^w(N)$, respectively.

Therefore, $P(N) = P^r(N) + P^w(N)$, $P^r(N) = \frac{IO^r(N)}{N/(DB)}$, $P^w(N) = \frac{IO^w(N)}{N/(DB)}$, and

$P(N) = \frac{(IO^r(N) + IO^w(N))}{N/(DB)}$. If a selection algorithm takes a constant number of passes (i.e., $P(N) = O(1)$), then the number of I/O operations done by the algorithm $IO(N) = O(N/(DB))$.

Definition 5

The rank of an element x in L_k is denoted by $Rank_x(k)$. The maximum possible rank of x in L_k is denoted as $Rank_x^u(k)$ and the minimum possible rank is denoted as $Rank_x^l(k)$. The j^{th} smallest in L_k is denoted by $E_k(j)$. Clearly, $t_i = E_0(i)$.

III. A New Algorithm

We have analyzed the algorithm of Deterministic Sampling Selection [24] and shown that it takes $(2 + \epsilon)$ passes under two assumptions as in Theorem 1.

Theorem 1

Selection on the PDS can be done in $(2 + \epsilon)$ passes using the Deterministic Sampling Selection (DSS) algorithm when $N = M^c$ for some constant c under the assumption that $|L_m| \approx M$.

In the DSS algorithm, to guarantee that t_i is always in the range $[u, v]$, the boundaries u and v cannot be too stringent. In other words, the interval $[u, v]$ has to be broad enough to bracket t_i . On the other hand, the number of elements in such an interval is also amplified

level by level exponentially in $\sqrt{M}$. Since even a single uncertainty in the rank of an element in L_k results in uncertainties of $\sqrt{M}$ in L_{k+1} . Thus, the uncertainties are accumulated over the levels of sampling and be exponential in $\sqrt{M}$, in the worst case.

In this section, we present a new algorithm, called *Recursive Sampling Selection (RSS)* algorithm, which takes $(2 + \varepsilon)$ passes with no assumptions (on N or $|L_m|$). *RSS* identifies an optimal interval in L_m to bracket t_i such that no amplification happens in the uncertainty of ranks of keys.

Definition 6

In L_k , the optimal boundaries $[u^*, v^*]$ for $E_{k-1}(i)$, where $E_{k-1}(i)$ is the i^{th} smallest element in L_{k-1} , are defined as follows: $\nexists [u', v']$, $u' > u^*$ and $v' < v^*$, that guarantee $E_{k-1}(i) \in [u', v']$.

Lemma 1

In L_k , the optimal boundaries for $E_{k-1}(i)$ are $u^* = E_k(i / \sqrt{M} - |L_k|/M)$ and $v^* = E_k(i / \sqrt{M})$. The number of elements of L_{k-1} that have values within the optimal boundaries is no more than $2|L_k| / \sqrt{M}$. In other words,

$$|\{x | x \in L_{k-1}, x \in [u^*, v^*]\}| \leq 2|L_k| / \sqrt{M}.$$

Proof— $Rank_{u^*}(k-1) \in [i - |L_k| / \sqrt{M}, i]$, and $Rank_{v^*}(k-1) \in [i, i + |L_k| / \sqrt{M}]$. So $E_{k-1}(i) \in [u^*, v^*]$, and the number of remaining elements $\in [u^*, v^*]$ is no more than $2|L_k| / \sqrt{M}$.

Algorithm 1 $RSS(E, i)$

Input: A set of elements E , where $N \Leftarrow |E|$

Output: The i^{th} smallest element t_i of E

if $N < M$ **then**

 perform an in-core selection and output

else

 group the elements of L_0 into groups of size M each;

 perform $\sqrt{M}$ -sampling on each group of L_0

 and hence get L_1 ;

$u \Leftarrow RSS(L_1, i / \sqrt{M} - |L_0|/M)$;

$v \leftarrow RSS(L_1, i/\sqrt{M});$

$E' \leftarrow \{x | x \in E, x \in [u, v]\};$

(write E' into disks if necessary);

$i' \leftarrow i - |\{x | x \in E, x < u\}|;$

Output $RSS(E', i')$

end if

RSS algorithm is shown in Algorithm 1. The optimal boundaries u^*, v^* are selected recursively so that the number of selections needed on $(L_1, L_2, L_3, \dots)$ is $(2, 4, 8, \dots)$. Although the number of selections needed for the optimal boundaries on each level increases exponentially, the number of elements on each level also decreases exponentially by a factor of $(1/\sqrt{M}, 1/M, 1/M\sqrt{M}, \dots)$. We show below that RSS is a 2-pass algorithm.

Analysis—The number of I/O operations performed by RSS is shown below.

$$\begin{aligned} IO^r(N) &= 2N/(DB) + 2IO^r(N/\sqrt{M}) + IO^r(2N/\sqrt{M}) \\ &= 2N/(DB) \times (1 + 4\sqrt{M} + 16/M + \dots) \\ &= \frac{2N/(DB)}{1 - 4/\sqrt{M}} \end{aligned}$$

For reads, $2N/(DB)$ corresponds to two-passes through the input: one for sampling L_1 and the other for applying the boundaries. $2IO^r(N/\sqrt{M})$ is for selection of the optimal boundaries. $IO^r(2N/\sqrt{M})$ is for the remaining no more than $2N/\sqrt{M}$ elements after applying the boundaries. For writes, the only difference is that $2N/(\sqrt{M}DB)$ is for writing the remaining no more than $2N/\sqrt{M}$ elements after bracketing.

$$\begin{aligned} IO^w(N) &= 2N/(\sqrt{M}DB) + 2IO^w(N/\sqrt{M}) \\ &+ IO^w(2N/\sqrt{M}) \\ &= \frac{2N/(\sqrt{M}DB)}{1 - 4/\sqrt{M}} \end{aligned}$$

Given $M \gg 4$, $IO(N) \approx 2N/(DB)$, so $P(N) \approx 2$.

Theorem 2

Selection on PDS can be done in very nearly 2 passes using the Recursive Sampling Selection (RSS) algorithm.

IV. QuickSelect on PDS

QuickSelect is an in-core selection algorithm with an expected running time of $\Theta(N)$ and a worst-case running time of $O(N^2)$. We have extended *QuickSelect* on the PDS and compared it with *DSS* and *RSS* algorithms. On PDS, *QuickSelect* works as following.

- Step 1 Read M elements into memory and pick a *pivot* value;
- Step 2 Partition the M elements into: $< pivot$, $pivot$, and $> pivot$ (write back to disks if necessary);
- Step 3 Continue reading and partitioning M elements until all the N elements are scanned;
- Step 4 Decide which partition t_i is in and update i if necessary;
- Step 5 Repeat Steps 1 to 4 on the right partition and new i until the partition is of size smaller than M .
- Step 6 Perform an in-core selection and output;

The best case happens when the *pivot* is the i^{th} smallest element. $IO_{best}^r(N) = N/(DB)$, $IO_{best}^w(N) = 0$, so $P_{best}(N) = 1$. However, the best case happens extremely rarely, since the probability that *pivot* = t_i is $1/N$. Also, *QuickSelect* suffers from the bad worst-case performance. $IO_{worst}^r(N) = N/(DB) + IO_{worst}^r(N-1) = O((N/(DB))^2)$, $IO_{worst}^w(N) = O((N/(DB))^2)$, so $P_{worst}(N) = O(N/(DB))$. On an average, assuming $i = N/2$,

$$IO_{avg}^r(N) = \frac{N}{DB} + \frac{2}{N} \sum_{j=\frac{N}{2}}^N IO_{avg}^r(j) > \frac{2N}{DB}$$

Similarly, $IO_{avg}^w(N) > 2N/(DB)$. So $P_{avg}(N) > 4$.

V. Experiments and Results

We have simulated the PDS and compared *Deterministic Sampling Algorithm (DSS)*, *Recursive Sampling Algorithm (RSS)*, and *QuickSelect*. In the simulated PDS, to perform $\sqrt{M}$ -sampling, a write buffer of size (DB) is kept in the memory for the $\sqrt{M}$ samples and the remaining $(M - DB)$ is used as the reading buffer to load elements from the disks. $\sqrt{M}$ -sampling is then done on the read buffer and the samples are kept in the write buffer. Once the write buffer is full, the $(DB) \sqrt{M}$ -samples are then written into D disks in parallel. In the *DSS* algorithm, we have used $\delta = (2c - 2 + \varepsilon) \sqrt{M} \times |L_m|/M$ and $\varepsilon = 0.005$. We have focused on the challenging instance of selection, namely selecting the median (i.e., $i = N/2$).

Three test data sets of size 100, 000, 1, 000, 000, and 10, 000, 000 were generated on uniform distribution (in the interval $[0, 1000]$) and normal distribution with $\mu = 500$, $\sigma = 100$. On the simulated PDS, $M = 10, 000$ and $B = 250$. We tried different numbers of disks. In particular, we have employed $D = 1$ (sequential), $D = 5$, and $D = 10$.

Since the results are similar for different sizes of data sets, we report the performance only on the largest data set of size 10, 000, 000. *I/O operations* and *passes* taken by the algorithms on uniform distribution and normal distribution are shown in Table I and Table II, respectively. The *RSS* and *DSS* algorithms are comparable, with similar $(2 + \varepsilon)$ passes, and outperform the *QuickSelect*. The number of passes taken by *QuickSelect* is three to six

times that of *RSS* or *DSS*. *QuickSelect* algorithm is slightly better on the normal distribution than on the uniform distribution, since on the normal distribution the probability of picking a pivot near the median is higher.

The parallelization of I/O operations on the PDS is excellent. For instance in Table I, the number of reads by *RSS* with five disks is $\frac{1}{4.99}$ of that with a single disk, and with ten disks it is $\frac{1}{9.96}$ of that on a single disk.

Since *DSS* is a $(2 + \epsilon)$ -pass algorithm under the assumption that $N = M^c$ for a constant c , the difference between *RSS* and *DSS* would be clearer if N is far greater than M . Instead of increasing the test data to a very large number, we have decreased M to ensure N is relatively much greater than M . Table III shows the results on 10,000,000 data on the uniform distribution with one disk. When M decreases from 1,000 to 250, in the *DSS* algorithm, the number of *passes* increases from 3.7 to nearly 10, while in *RSS* it is still around 2.3 to 2.6, indicating the robustness of the proposed algorithm *RSS*.

VI. Discussion and Conclusions

In this paper, we have proposed a $(2+\epsilon)$ -pass deterministic algorithm called *RSS* for exact out-of-core selection. To the best of our knowledge, no other two-pass exact algorithm on out-of-core selection has been proposed. Our analysis of the *DSS* algorithm shows that it is also a $(2 + \epsilon)$ -pass algorithm under the following assumptions: 1) $N = M^c$ for some constant c , and 2) $|L_m| \approx M$ [24]. Our proposed algorithm achieves $(2+\epsilon)$ -passes with no such assumptions.

We have extended the *QuickSelect* algorithm on the PDS. Programs simulating the PDS have been used to compare these algorithms. Our experimental results show that *DSS* and *RSS* outperform the *QuickSelect* algorithm. *DSS* and *RSS* are comparable to each other when $N = M^c$ for a small constant c . When $N \gg M$, the proposed algorithm *RSS* is still maintaining two passes, while *DSS* algorithm takes a rapidly increasing number of passes. Both theoretical analysis and experimental results show that the proposed $(2+\epsilon)$ -pass exact algorithm *RSS* for out-of-core selection is robust and promising.

Acknowledgments

This work has been supported in part by the following grants: NSF 0829916 and NIH R01-LM010101.

References

1. Gray J, Chaudhuri S, Bosworth A, Layman A, Reichart D, Venkatrao M, Pellow F, Pirahesh H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min Knowl Discov*. Jan; 1997 1(1):29–53.
2. Davies E. Fast implementation of generalised median filter. *Electronics Letters*. 2007; 43(9):505–507.
3. Tyler DE. Robust statistics: Theory and methods. ricardo a. maronna, r. douglas martin, and victor j. yohai. *Journal of the American Statistical Association*. Jun.2008 103:888–889.

4. Rousseeuw, P.J.; Leroy, A.M. Robust regression and outlier detection. New York, NY, USA: John Wiley & Sons, Inc.; 1987.
5. Blum M, Floyd RW, Pratt V, Rivest RL, Tarjan RE. Time bounds for selection. J Comput Syst Sci. Aug; 1973 7(4):448–461.
6. Floyd RW, Rivest RL. Expected time bounds for selection. Commun ACM. Mar; 1975 18(3):165–172.
7. Reischuk R. Probabilistic parallel algorithms for sorting and selection. SIAM Journal on Computing. 1985; 14(2):396–409.
8. Megiddo N. Probabilistic parallel algorithms for sorting and selection. Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University. Oct.1925
9. Rajasekaran, S.; Sen, S. Random sampling techniques and parallel algorithms design. In: Reif, J., editor. Synthesis of Parallel Algorithms. San Mateo, California: Morgan-Kaufmann Publishers; 1993. p. 411-451.
10. Rajasekaran, S.; Wei, D. Selection, routing, and sorting on the star graph. Parallel Processing Symposium 1993 Proceedings of Seventh International; apr 1993; p. 661-665.
11. Rajasekaran S. Randomized selection on the hypercube. Journal of Parallel and Distributed Computing. 1996; 37(2):187–193.
12. Krizanc, D.; Narayanan, L. Optimal algorithms for selection on a mesh-connected processor array. Parallel and Distributed Processing, 1992 Proceedings of the Fourth IEEE Symposium on; dec 1992; p. 70-76.
13. Rajasekaran S. Mesh connected computers with fixed and reconfigurable buses: packet routing and sorting. Computers IEEE Transactions on. May; 1996 45(5):529–539.
14. Rajasekaran, S.; Reif, J. Derivation of randomized sorting and selection algorithms. In: Paige, R.; Reif, J.; Watcher, R.; Ismail, M., editors. Parallel Algorithm Derivation and Program Transformation. Vol. 231. Springer; US: 1993. p. 187-205.ser. The Springer International Series in Engineering and Computer Science
15. Rajasekaran S. Sorting and selection on interconnection networks. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. 1995:275–296.
16. Krizanc, D.; Narayanan, L. Multi-packet selection on mesh-connected processor arrays. Parallel Processing Symposium 1992 Proceedings Sixth International; mar 1992; p. 602-605.
17. Plaxton, CG. PhD dissertation. Stanford, CA, USA: 1989. Efficient computation on sparse interconnection networks. aAI9011556
18. Rajasekaran S, Chen W, Yooseph S. Unifying themes for selection on any network. Journal of Parallel and Distributed Computing. 1997; 46(1):105–111.
19. Fu L, Rajasekaran S. Evaluating holistic aggregators efficiently for very large datasets. The VLDB Journal. May; 2004 13(2):148–161.
20. Munro, J.I.; Paterson, M.S. Selection and sorting with limited storage. Proceedings of the 19th Annual Symposium on Foundations of Computer Science, ser. SFCS '78; Washington, DC, USA: IEEE Computer Society; 1978. p. 253-258.
21. Alsabti, K.; Ranka, S.; Singh, V. A one-pass algorithm for accurately estimating quantiles for disk-resident data. Proceedings of the 23rd International Conference on Very Large Data Bases, ser. VLDB '97; San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1997. p. 346-355.
22. Manku GS, Rajagopalan S, Lindsay BG. Approximate medians and other quantiles in one pass and with limited memory. SIGMOD Rec. Jun; 1998 27(2):426–435.
23. Vitter J, Shriver E. Algorithms for parallel memory, i: Two-level memories. Algorithmica. 1994; 12:110–147.
24. Rajasekaran, S. Selection algorithms for parallel disk systems. High Performance Computing, 1998. HIPC '98. 5th International Conference On; dec 1998; p. 343-350.

Figure 1.
Deterministic Sampling

Figure 2.
Illustration of Observation 2

Table I

I/O Performance under Uniform Distribution

		D=1		D=5		D=10	
		#I/O	#Pass	#I/O	#Pass	#I/O	#Pass
RSS	read	82065	2.05	16436	2.05	8237	2.06
	write	838	0.02	181	0.02	100	0.03
	total	82903	2.07	16617	2.08	8337	2.08
DSS	read	82905	2.07	16631	2.08	8265	2.07
	write	1673	0.04	361	0.05	156	0.04
	total	84578	2.11	16992	2.12	8421	2.11
QuickSelect	read	181800	4.55	30624	3.83	18639	4.66
	write	190480	4.76	39028	4.88	28504	7.13
	total	372280	9.31	69652	8.71	47143	11.79

Table II

I/O Performance under Normal Distribution

		D=1		D=5		D=10	
		#I/O	#Pass	#I/O	#Pass	#I/O	#Pass
RSS	read	82069	2.05	16438	2.05	8237	2.06
	write	840	0.02	182	0.02	100	0.03
	total	82909	2.07	16620	2.08	8337	2.08
DSS	read	82877	2.07	16533	2.07	85001	2.13
	write	1658	0.04	312	0.04	274	0.07
	total	84535	2.11	16845	2.11	8774	2.19
QuickSelect	read	116966	2.92	24463	3.06	17819	4.45
	write	122909	3.07	31248	3.91	27296	6.82
	total	239875	6.00	55711	6.96	45115	11.28

Table IIIComparison of *RSS* vs. *DSS*

	<i>RSS</i> (#Pass)	<i>DSS</i> (#Pass)
M=1,000, B = 100		
read	2.18268	3.10264
write	0.07313	0.5768
total	2.25581	3.67944
M=500, B = 100		
read	2.2676	3.40156
write	0.106305	0.74037
total	2.373905	4.14193
M=250, B = 50		
read	2.43405	6.97569
write	0.172205	2.60818
total	2.606255	9.58387