Multiple String Matching Algorithms Performance Study on Beowulf Clusters

Prasad J C
Research Scholar
Dr M G R University,
TamilNadu

Dr. K S M Panicker
Professor, CSE Dept.
FISAT, Angamaly
Cochin, India

ABSTRACT
Efficiency of multiple string searching has become more relevant with the large and redundant amount of data. The size of storage devices has increased in terms of Terabytes and modern processors are capable to perform parallel computation with multi-core architecture. Beowulf cluster architecture is considered for parallel computations, in which 40 nodes and two quad core processor servers perform multiple pattern searching operations with different algorithms. Multiple pattern searching is essential for intrusion detection systems (IDS), which has the ability to search through packets and identify content that matches known attacks. Latest advancements in DNA sequencing, web search engines, database operations, signal processing, error detection, speech and pattern recognition areas require multiple patterns searching problem to process terabytes of data. Space and time efficient string matching algorithms are therefore important for this purpose.

General Terms
Aho-Corasick algorithm, Wu-Manber algorithms, AC-Bitmap and q-Grams algorithms

Keywords
Beowulf cluster, Multiple string matching algorithm performance, MPI Programming.

1. INTRODUCTION
Efficiency of multiple string searching has become more relevant with the large and redundant amount of data. The size of storage devices has increased in terms of Terabytes and modern processors are capable to perform parallel computation with multi-core architecture. Beowulf cluster architecture is considered for parallel computations, in which 40 nodes and two quad core processor servers perform multiple pattern searching operations with different algorithms. Multiple pattern searching is essential for intrusion detection systems (IDS), which has the ability to search through packets and identify content that matches known attacks. Latest advancements in DNA sequencing, web search engines, database operations, signal processing, error detection, speech and pattern recognition areas require multiple patterns searching problem to process terabytes of data. Space and time efficient string matching algorithms are therefore important for this purpose.

2. RELATED WORK
The naive approach to multi pattern approximate searching is to perform r separate searches, one per pattern. If we use the classical O(mn) algorithm, the time is O(rmn), where m is the length of the pattern and n is the length of the text to be searched [1].

The performance of signature-based NIDS is dominated by the speed of string matching algorithm used to compare the packet header and payload with signatures. For instance, Snort, an open source NIDS, takes over 2,500 patterns as signatures and spends more than 80% of CPU time on string matching. A NIDS claims as a fast string matching algorithm to reduce its load. Otherwise, an underperforming system not only becomes the network bottleneck, but also misses some critical attacks [2].

The famous Aho-Corasick (AC) algorithm is an automaton-based algorithm that guarantees the linear time complexity in worst case situation. There are two kinds of data structure used in AC algorithm: non-deterministic finite state automaton (NFA) and deterministic finite state automaton (DFA). When DFA is adopted, for every state, every symbol has the corresponding next state. Hence for each input character, only single memory reference is needed, and the time-complexity is guaranteed to be O(n), where n is the length of input string. However, the demanded memory space is large. On the other hand, when NFA is adopted, for every state, not every symbol has the corresponding next state. In this method, failure path is needed, and time-complexity then becomes O(n + k), where k is the number of times which failure path is taken. [3]

Wu-Manber algorithms can support tens of thousands of patterns and is better than agrep. The design of the algorithm concentrates on typical searches rather than on worst-case behavior. This allows us to make some engineering decisions that believing are crucial for making the algorithm significantly faster than other algorithms in practice. Instead of looking at characters from the text one by one, we consider them in blocks of size B. During the scanning stage, compute a hash value h based on the current B characters from the text. Expected running time of this algorithm is less than linear in the size of the text. Unless the patterns are very small or there are very few of them, this algorithm is significantly faster. The original egrep and fgrep could not handle (or took too long for) more than few hundreds patterns. [4]

AC-Bitmap: N. Tuck, T. Sherwood, B. Calder, G. Varghese proposed a modified AC algorithm by applying IP routing lookup techniques. According to the form of the NFA in the AC algorithm, they used bitmaps that correspond to symbols to record the state transition of the non failure path. In this way, every node in the finite automaton only uses a pointer pointing to the next state list instead of allocating all the pointers to the next state. Thus, AC-Bitmap can decrease a great deal of the demanded memory for implementing NFA.

Multi-pattern string matching with q-Grams: L. Salmela, J. Tarhio and J. Kytojoki proposed Multi-pattern string matching with q-grams in the year 2007. Given a text position, a filter can tell if there cannot be a match at this position using filtering approach (eg: hashing function). A good filter is fast and produces few false positives. Verification to filtering is used to distinguish between false and true positives. Filtering approach to multiple pattern

matching transform patterns to sequences of q-grams and filter with a character class pattern built from the transformed pattern set. Then it verify with a Rabin-Karp style algorithm.[6] q-Grams filters have three, recognition zones` depending on the number of errors : Guarantee zone (finds all approximate matches), Heuristic zone (finds some of the approximate matches) and Negative zone (guaranteed not to find matches). Behavior in the Heuristic Zone is hard to predict.

3. MULTIPLE PATTERNS SEARCHING PROBLEM

Given a pattern set P and a text T, report all occurrences of all the patterns in the text. The text T is drawn from the alphabet Σ (of size σ). The pattern set, P is a set of r patterns each of which is a string of characters over the alphabet Σ. [Assume that all patterns have the same length m (for simplicity of description); multiple pattern search algorithms do not make such assumption]. A multiple string pattern matching problem can be defined as follows. Let T be a large text of n number of character size and P be set of r pattern of length m1, m2, …… mr. The characters of Text T, and r patterns stored in arrays as T[1] ,T[2] ,….,T[n] and Pattern P = P[1,1] , P[1,2]….P[1,m1], P[2,1] , P[2,2], ….P[2,m2] , ... P[r, 1], P[r, 2],…….,P[r, mr]. The characters of both T and P belong to a finite set of elements of the set S and m1, m2, ……mr << n. Each searching processes is done with preprocessing and searching stage. Preprocessing stage computes the hash function (or any signature method) of each pattern and store them. During Searching stage, for each text position i compute signature function and search for the signature value from the saved values of the patterns, and identify all the occurrence of the pattern P in text T. Two types of input data have considered (natural language input string and DNA sequence string) for the evaluation of algorithms. The actual task of searching is done parallel among the processors from 0 to p-1. [7]

Master node decomposes the text into r subtexts and distributed to available workers (nodes)[6]. Each subtext contains k = [(n-mj+1) / r] + mj-1, characters, where k is the successive characters of the complete text. The value of j ranges from 1 to r. There is an overlap of m-1 successive characters between successive sub texts for different pattern values of mj. So there will be a redundancy of [r(m1+m2+ ……+mr)-r] characters for processing. The objective is to compare the result of searching with different algorithms. So redundancy of searching does not have relevance in the system[8].

4. EXPERIMENTAL SETUP

Beowulf based systems - Dhakshina Cluster Series-I and Series-II are used for the experiment. During HPL benchmarking[9], the speed of Dakshina-II has recorded 9600 cores floating operations in a second where as Dakshina-I performed 7000 cores floating point operations per second. Two servers of IBM X series Quad Core Xeon processors used separately for Network Information Service and Job Scheduling. Linux kernel used with configuration 1 is customized Debian of 2.6.18 whereas for configuration 2 is of the version 2.6.26. The configuration of server is: 146 GB SAS HDD (Serial Attached SCSI (SAS), the logical evolution that satisfies the enterprise data center requirement of scalability, performance, reliability and manageability, while leveraging a common electrical and physical connection interface with Serial ATA (SATA). This compatibility provides users with unprecedented choices for server and storage subsystem deployment. 2 GB RAMNIC 2G (Giga byte Ethernet Card) is used. 32 computational nodes of Pentium 4 HT machines and other 3 PCs as login nodes used with this system.

Network Configuration involves with the following features. Realtek 8169 Gigabit network card on each compute nodes. The Realtek RTL8169SB(L) NIC Gigabit ethernet controllers (RTL8169SB (128 QFP) & RTL8169SBL (128 LQFP)) combine a triple-speed IEEE 802.3 compliant Media Access Controller (MAC) with a triple-speed ethernet transceiver, 32-bit PCI bus controller, and embedded memory. Functions such as crossover detection and auto-correction, polarity correction, adaptive equalization, cross-talk cancellation, echo cancellation, timing recovery, and error correction are implemented to provide robust transmission and reception capability at high speeds. The versatility of the LWAKE pin provides motherboards with Wake-On-LAN (WOL) functionality. Broadcom NetExtreme Gigabit ethernet card X 2 is the Ethernet card in the master nodes which represent the world's first support for the PCI Express specification; the BCM5721 combines innovative performance enhancements with a PCI Express 1x host interface enabling 2 Gbps throughput for the most demanding server applications. To reduce the total cost of server ownership, the BCM5721 supports the Intelligent Platform Management Interface (IPMI) 1.5 manageability standard that allows servers to be remotely managed. The BCM5721 also incorporates the industry's most advanced server software (the Broadcom Advanced Server Program or BASP) which offers several innovative teaming functions not available from any other GbE solution. Our system used the ProCurve Switch 1400 series provides plug-and-play simplicity for high-bandwidth connectivity, with a fan-less design that promotes silent operation. The 1400-24G has 22 10/100/1000 ports and two dual-personality ports for 10/100/1000 or mini-GBIC connectivity.[10]

Message Passing Interface (MPI) is a language independent communications protocol used to program parallel computers. Both point-to-point and collective communication are supported. MPI's goals are high performance, scalability, and portability. MPI remains the dominant model used in high-performance computing today. Most MPI implementations consist of a specific set of routines (i.e., an API) callable from FORTRAN, C, or C++ and from any language capable of interfacing with such routine libraries. [11][12]

5. RESULT

To get a reliable and consistent performance result, the average of ten executions for multiple patterns of constant length is given in the table. The results of sequential and parallel implementation of Aho-Corasick NFA algorithm, Aho-Corasick DFA algorithm, Wu-Manber algorithms, AC-Bitmap algorithm, and Multi-pattern string matching algorithms with q-grams are shown in table.

Table 1: Aho-Corasick NFA Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in sequential Manner

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	10.2	30.68	44.89	60.24	72.45
	m_2=10	13.3	40.93	59.10	78.97	94.05
	m_3=15	14.2	42.24	63.07	83.58	100.93
	m_4=20	16.4	49.38	72.58	96.84	116.57
	m_5=25	17.6	52.73	78.05	103.4	124.38
	m_6=50	20.1	60.22	88.14	117.2	142.34
	m_7=100	24.9	74.28	110.0	146.8	176.25

	m_8=250	29.7	89.61	131.5	174.9	210.58
	m_9=500	32.2	96.78	142.3	189.4	228.07
	m_{10}= 750	34.5	103.8	152.9	202.6	244.15
	m_{11}=1000	36.7	110.4	162.7	215.7	260.22
	m_{12}=2000	39.1	117.1	173.4	229.8	277.45
	m_{13}=5000	44.3	133.3	196.7	259.1	313.77
	m_{14}=10000	45.2	135.6	201.3	265.4	320.67

Table 2: Aho-Corasick NFA Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in Dakshina-I with 5 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m1=5	3.42	10.26	15.13	20.93	24.17
	m2=10	4.44	13.25	19.76	26.88	31.72
	m3=15	4.77	14.33	21.88	27.67	33.46
	m4=20	5.47	16.25	24.44	32.77	38.34
	m5=25	5.77	17.22	25.87	34.01	41.53
	m6=50	6.77	20.22	29.83	39.82	47.96
	m7=100	8.13	24.35	36.27	48.57	58.24
	m8=250	9.95	29.43	43.66	58.02	70.67
	m9=500	10.4	32.17	47.17	63.23	76.05
	m10= 750	11.2	34.18	51.03	67.23	81.56
	m11=1000	12.6	36.87	54.33	71.49	86.23
	m12=2000	13.2	39.48	57.21	76.12	92.87
	m13=5000	14.9	44.26	65.29	86.13	104.45
	m14=10000	15.0	45.36	66.98	88.29	106.73

Table 3: Aho-Corasick NFA Multiple Pattern Search Algorithm Execution Time in Second for a file size 60MB in Dakshina-I with 10 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m1=5	2.12	6.73	10.83	13.45	16.23
	m2=10	2.98	8.93	13.34	17.44	20.99
	m3=15	3.15	9.92	14.32	18.83	22.64
	m4=20	3.74	10.9	16.16	21.44	25.94
	m5=25	3.96	11.8	17.44	22.82	27.85
	m6=50	4.51	13.5	19.48	26.22	31.82
	m7=100	5.73	16.7	24.67	32.13	39.21
	m8=250	6.81	19.7	29.25	38.76	46.75
	m9=500	7.71	21.1	31.24	42.07	50.67
	m10= 750	7.72	23.0	33.99	45.52	54.36
	m11=1000	8.22	24.6	36.24	47.97	57.92
	m12=2000	8.77	26.3	38.56	51.34	61.34
	m13=5000	9.87	29.6	43.64	57.77	69.57
	m14=10000	10.0	30.2	44.65	58.97	71.45

Table 4: Aho-Corasick NFA Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in Dakshina-II with 5 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m1=5	2.82	8.46	12.67	16.42	19.57
	m2=10	3.67	11.03	16.28	21.67	25.89
	m3=15	3.98	11.96	17.44	22.98	27.96
	m4=20	4.56	13.58	19.99	26.54	31.98
	m5=25	4.95	14.62	21.56	28.95	34.39
	m6=50	5.52	16.56	24.44	32.53	39.09
	m7=100	6.89	20.76	30.43	40.22	48.34
	m8=250	8.24	24.86	36.08	47.90	57.72
	m9=500	8.94	26.65	39.28	51.89	62.66
	m10= 750	9.84	28.40	41.92	55.78	67.29
	m11=1000	10.0	30.22	44.94	59.23	71.45
	m12=2000	10.7	32.22	47.35	62.99	76.09
	m13=5000	12.3	36.85	53.98	71.85	86.08
	m14=10000	12.4	37.23	54.78	72.77	87.82

Table 5: Aho-Corasick NFA Multiple Pattern Search Algorithm Execution Time (sec) for a file size 60 MB in Dakshina-II with 10 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	1.87	5.61	8.28	11.03	13.88
	m_2=10	2.44	7.32	10.80	14.23	17.22
	m_3=15	2.65	7.78	11.51	15.22	18.76
	m_4=20	3.03	9.05	13.40	17.63	21.33
	m_5=25	3.23	9.78	14.67	18.95	22.93
	m_6=50	3.66	11.34	16.46	21.66	26.09
	m_7=100	4.65	13.86	20.66	26.24	32.56
	m_8=250	5.22	16.20	24.56	31.32	38.56
	m_9=500	5.87	17.34	26.34	34.78	41.50
	m_{10}= 750	6.32	18.67	27.45	37.98	44.28
	m_{11}=1000	6.73	20.22	29.45	39.44	47.23
	m_{12}=2000	7.35	21.67	31.95	41.44	50.82
	m_{13}=5000	8.87	24.23	35.78	47.25	57.79
	m_{14}=10000	8.88	24.97	36.98	48.87	58.23

Table 6: Wu-Manbar Multiple Pattern Search Algorithm Execution Time (Sec) for a file size 60 MB in sequential Manner

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	3.44	10.33	14.44	20.44	24.22
	m_2=10	4.44	13.66	19.77	26.44	31.67
	m_3=15	4.77	14.88	21.22	27.88	33.68
	m_4=20	5.55	16.74	24.48	32.48	38.97
	m_5=25	5.95	17.65	26.30	34.85	41.55
	m_6=50	6.77	20.73	29.67	39.82	47.54
	m_7=100	8.45	24.87	36.56	48.85	58.69
	m_8=250	9.93	29.89	43.88	58.23	70.78
	m_9=500	10.77	32.33	47.54	63.43	76.04
	m_{10}= 750	11.44	34.66	51.22	67.66	81.48
	m_{11}=1000	12.27	36.44	54.55	71.89	86.55
	m_{12}=2000	13.45	39.25	57.99	76.26	92.84
	m_{13}=5000	14.77	44.58	65.88	86.67	104.76
	m_{14}=7500	15.78	47.32	68.45	89.98	107.87
	m_{15}=10000	14.23	44.12	64.55	86.33	104.78

Table 7: Wu-Manbar Multiple Pattern Search Algorithm Execution Time (Sec) for a file size 60 MB in Dhakshina Cluster-I with 5 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m1=5	1.11	3.24	4.82	6.54	7.46
	m2=10	1.44	4.33	6.13	8.34	10.04
	m3=15	1.52	4.55	6.77	8.89	10.87
	m4=20	1.77	5.25	7.77	10.36	12.49
	m5=25	1.85	5.66	8.22	11.05	13.21
	m6=50	2.11	6.44	9.39	12.33	15.05
	m7=100	2.78	7.87	11.72	15.76	18.78
	m8=250	3.16	9.55	14.40	18.67	22.55
	m9=500	3.45	10.3	15.22	20.24	24.33
	m10= 750	3.69	11.1	16.43	21.34	26.34
	m11=1000	3.94	11.8	17.43	23.08	27.87
	m12=2000	4.22	12.6	18.67	24.65	29.59
	m13=5000	4.74	14.2	21.23	27.89	33.56
	m14=7500	5.08	15.2	21.94	28.78	34.24
	m15=10000	4.57	14.2	20.65	27.66	33.23

Table 8: Wu-Manbar Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in Dhakshina Cluster-I with 10 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	0.75	2.23	3.30	4.44	5.33
	m_2=10	.98	3.05	4.45	5.88	6.99
	m_3=15	1.09	3.23	4.77	6.33	7.55
	m_4=20	1.25	3.69	5.44	7.27	8.84
	m_5=25	1.33	3.95	5.84	7.71	9.32
	m_6=50	1.53	4.52	6.55	8.78	10.65
	m_7=100	1.88	5.52	8.17	10.92	13.02
	m_8=250	2.22	6.72	9.92	12.91	15.54
	m_9=500	2.40	7.25	10.62	14.23	16.92
	m_{10}= 750	2.62	7.66	11.35	15.05	18.22
	m_{11}=1000	2.77	8.38	12.01	15.88	19.22
	m_{12}=2000	2.89	8.73	12.70	17.23	20.53
	m_{13}=5000	3.29	9.81	14.51	19.22	23.15
	m_{14}=7500	3.55	10.4	15.22	20.22	23.76
	m_{15}=10000	3.22	9.29	14.23	19.56	23.34

Table 9: Wu-Manbar Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in Dhakshina Cluster-II with 5 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	.90	2.70	3.94	5.29	6.36
	m_2=10	1.17	3.60	5.20	6.96	8.29
	m_3=15	1.26	3.74	5.59	7.36	8.89
	m_4=20	1.46	4.36	6.41	8.52	10.25
	m_5=25	1.56	4.66	6.88	9.11	10.95
	m_6=50	1.77	5.28	7.77	10.4	12.6
	m_7=100	2.21	6.53	9.76	12.82	15.51
	m_8=250	2.66	7.89	11.69	15.38	18.61
	m_9=500	2.91	8.62	12.62	16.75	20.05
	m_{10}= 750	3.05	9.21	13.46	17.91	21.53
	m_{11}=1000	3.24	9.72	14.33	18.98	22.91
	m_{12}=2000	3.47	10.3	15.33	20.22	24.42
	m_{13}=5000	3.91	11.8	17.34	22.82	27.62
	m_{14}=7500	4.17	12.4	18.06	23.76	28.41
	m_{15}=10000	3.78	11.6	17.11	22.81	27.67

Table 10: Wu-Manbar Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in Dhakshina Cluster-II with 10 nodes.

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m1=5	.66	1.88	2.76	3.69	4.41
	m2=10	.83	2.51	3.62	4.83	5.76
	m3=15	.89	2.59	3.86	5.13	6.17
	m4=20	1	3.04	4.45	5.93	7.14
	m5=25	1.05	3.13	4.77	6.35	7.65
	m6=50	1.25	3.69	5.42	7.19	8.73
	m7=100	1.51	4.52	6.72	8.97	10.76
	m8=250	1.83	5.48	8.05	10.71	12.89
	m9=500	1.99	5.94	8.72	11.62	13.45
	m10= 750	2.13	6.35	9.36	12.42	14.91
	m11=1000	2.21	6.71	9.88	13.11	15.81
	m12=2000	2.38	7.16	10.59	14.08	17.03
	m13=5000	2.71	8.14	11.98	15.85	19.15
	m14=7500	2.90	8.68	12.59	16.49	19.72
	m15=10000	2.62	8.09	11.88	15.81	19.22

Table 11: AC-Bitmap Multiple Pattern Search Algorithm Execution Time in Seconds for a file size 60 MB in sequential manner.

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m1=5	13.2	34.0	48.02	63.76	75.87
	m2=10	16.5	44.2	62.98	82.15	97.23
	m3=15	17.7	45.6	66.27	86.93	104.2
	m4=20	19.3	52.4	76.67	99.94	120.0
	m5=25	20.8	55.9	82.65	107.0	127.3
	m6=50	24.1	63.4	91.54	120.5	146.1
	m7=100	27.9	77.5	113.8	149.5	179.7
	m8=250	32.8	92.7	134.9	178.0	213.9
	m9=500	35.7	99.9	145.9	193.1	231.2
	m10= 750	37.7	106.	156.0	206.6	247.5
	m11=1000	39.9	113.	165.9	218.9	263.7
	m12=2000	42.4	119.	176.8	232.4	279.8
	m13=5000	47.2	136.	199.2	263.1	317.2
	m14=10000	48.4	139.	204.5	268.5	323.8

Table 12: AC-Bitmap Multiple Pattern Search Algorithm Execution Time (Second) for a file size 60 MB in Dakshina-I with 5 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	4.53	11.5	16.34	21.56	25.63
	m_2=10	5.78	15.0	21.56	27.83	32.95
	m_3=15	6.03	15.8	22.76	29.46	34.78
	m_4=20	6.65	17.4	25.86	33.68	40.58
	m_5=25	7.23	18.4	27.45	35.84	42.92
	m_6=50	8.10	21.3	30.89	40.23	49.34
	m_7=100	9.29	26.3	38.81	50.29	60.28
	m_8=250	11.2	31.4	45.37	59.24	71.93
	m_9=500	12.8	33.8	49.05	64.75	77.58
	m_{10}= 750	12.7	35.7	52.22	69.22	83.32
	m_{11}=1000	13.6	38.7	55.97	73.67	88.33
	m_{12}=2000	14.2	40.2	59.45	78.03	93.67
	m_{13}=5000	15.7	45.7	66.71	88.64	106.7
	m_{14}=10000	16.4	47.2	68.87	90.63	108.7

Table 13: AC-Bitmap Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in Dakshina-I with 10 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	2.98	7.71	10.83	14.72	16.87
	m_2=10	3.77	9.93	14.23	18.73	21.46
	m_3=15	3.99	10.2	14.87	19.57	21.79
	m_4=20	4.37	11.7	17.31	22.19	26.77
	m_5=25	4.64	12.4	18.39	23.81	28.31
	m_6=50	5.42	14.2	20.37	26.97	32.95
	m_7=100	6.33	17.3	25.22	33.21	39.97
	m_8=250	7.33	20.6	29.91	39.83	47.79
	m_9=500	8.01	22.4	32.39	42.92	51.38
	m_{10}= 750	8.47	23.9	34.77	46.04	55.07
	m_{11}=1000	8.91	25.4	36.93	48.77	58.79
	m_{12}=2000	9.17	26.9	39.57	51.28	62.42
	m_{13}=5000	10.7	30.6	44.43	58.47	70.53
	m_{14}=10000	10.9	31.4	45.82	59.81	72.37

Table 14: AC-Bitmap Multiple Pattern Search Algorithm Execution Time(Second) for a file size 60MB in Dakshina-II with 5 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	3.73	9.48	13.29	17.74	21.06
	m_2=10	4.58	12.5	17.36	22.71	26.48
	m_3=15	4.74	12.5	18.39	24.27	28.93
	m_4=20	5.42	14.6	21.43	27.87	33.43
	m_5=25	5.82	15.6	22.91	29.34	35.59
	m_6=50	6.72	17.4	25.32	33.57	40.23
	m_7=100	7.91	21.6	31.56	41.42	49.89
	m_8=250	9.03	25.4	37.19	49.83	59.48
	m_9=500	9.97	27.7	40.38	53.57	63.92
	m_{10}= 750	10.5	29.6	43.42	57.38	68.49
	m_{11}=1000	11.2	31.6	45.91	60.63	72.96
	m_{12}=2000	11.8	33.2	48.99	64.18	77.46
	m_{13}=5000	13.5	37.8	55.25	72.95	87.79
	m_{14}=10000	13.6	38.8	56.77	74.69	89.72

Table 15: AC-Bitmap Multiple Pattern Search Algorithm Execution Time(Second) for a file size 60 MB) in Dakshina-II with 10 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	2.44	6.21	8.81	11.68	13.92
	m_2=10	3.04	8.23	11.45	15.09	17.82
	m_3=15	3.26	8.37	12.14	15.94	19.09
	m_4=20	3.58	9.62	14.06	18.32	21.98
	m_5=25	3.85	10.2	15.15	19.59	23.33
	m_6=50	4.44	11.6	16.77	22.08	26.78
	m_7=100	5.14	14.2	20.85	27.38	32.91
	m_8=250	6.04	16.9	24.72	32.61	39.18
	m_9=500	6.57	18.3	26.72	35.37	42.33
	m_{10}= 750	6.93	19.5	28.61	37.81	45.31
	m_{11}=1000	7.33	20.8	30.39	40.07	48.31
	m_{12}=2000	7.77	21.9	32.37	42.53	51.22
	m_{13}=5000	8.66	25.0	36.47	48.16	58.05
	m_{14}=10000	8.89	25.6	37.43	49.15	59.27

Table 16: q-Grams Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in sequential Manner

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m1=5	3.56	10.56	15.42	20.45	24.72
	m2=10	4.46	13.84	19.85	26.44	31.67
	m3=15	4.33	14.34	21.22	27.92	33.98
	m4=20	5.62	16.74	24.75	32.49	38.91
	m5=25	5.93	17.63	26.04	34.52	41.73
	m6=50	6.84	19.07	29.02	38.76	47.04
	m7=100	8.52	24.22	36.21	48.23	58.15
	m8=250	9.45	29.65	43.72	58.11	69.24
	m9=500	10.24	32.02	47.03	63.04	76.88
	m10= 750	11.82	34.82	51.24	69.34	81.88
	m11=1000	11.26	34.91	53.23	70.94	84.63
	m12=2000	12.45	38.75	55.34	74.82	90.33
	m13=5000	13.47	43.42	64.34	84.43	103.4
	m14=1000	14.68	44.32	66.32	86.08	104.6

Table 17: q-Grams Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in Dhakshina Cluster-I with 5 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	**m_1=5**	1.14	3.32	4.98	6.54	7.92
	m_2=10	1.49	4.49	6.32	8.49	10.11
	m_3=15	1.45	4.64	6.83	8.97	10.89
	m_4=20	1.87	5.39	7.98	10.46	12.49
	m_5=25	1.96	5.69	8.37	11.15	13.39
	m_6=50	2.24	6.18	9.36	12.47	15.12
	m_7=100	2.79	7.79	11.65	15.48	18.68
	m_8=250	3.08	9.56	14.08	18.68	22.19
	m_9=500	3.32	10.29	15.11	20.25	24.67
	m_{10}= 750	3.79	11.19	16.46	22.26	26.28
	m_{11}=1000	3.64	11.22	17.09	22.78	27.17
	m_{12}=2000	4.08	12.48	17.79	23.98	28.98
	m_{13}=5000	4.38	13.93	20.67	27.09	33.18
	m_{14}=10000	4.73	14.19	21.28	27.54	33.55

Table 18: q-Grams Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in Dhakshina Cluster-I with 10 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	0.78	2.33	3.45	4.56	5.44
	m_2=10	0.94	3.11	4.45	5.91	7.14
	m_3=15	0.99	3.22	4.74	6.24	7.57
	m_4=20	1.28	3.76	5.55	7.27	8.66
	m_5=25	1.35	3.97	5.83	7.71	9.32
	m_6=50	1.56	4.29	6.47	8.65	10.48
	m_7=100	1.93	5.42	8.08	10.77	12.96
	m_8=250	2.14	6.63	9.78	12.94	15.43
	m_9=500	2.33	7.15	10.49	14.06	17.13
	m_{10}= 750	2.67	7.79	11.45	15.47	18.25
	m_{11}=1000	2.54	7.79	11.87	15.79	18.85
	m_{12}=2000	2.79	8.65	12.34	16.67	20.09
	m_{13}=5000	3.06	9.71	14.37	18.81	23.06
	m_{14}=10000	3.05	9.95	14.83	19.19	23.29

Table 19: q-Grams Multiple Pattern Search Algorithm Execution Time in Second for a file size 60MB in Dhakshina Cluster-II with 5 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	0.96	2.79	4.09	5.43	6.56
	m_2=10	1.23	3.69	5.27	7.03	8.41
	m_3=15	1.18	3.83	5.65	7.41	8.99
	m_4=20	1.54	4.46	6.57	8.59	10.28
	m_5=25	1.59	4.68	6.82	9.13	11.06
	m_6=50	1.82	5.07	7.64	10.29	12.40
	m_7=100	2.28	6.38	9.57	12.73	15.36
	m_8=250	2.54	7.87	11.54	15.34	18.28
	m_9=500	2.75	8.48	12.43	16.66	20.29
	m_{10}= 750	3.15	9.19	13.50	18.29	21.63
	m_{11}=1000	2.99	9.25	14.08	18.74	22.36
	m_{12}=2000	3.33	10.2	14.64	19.77	23.81
	m_{13}=5000	3.58	11.4	16.96	22.23	27.24
	m_{14}=1000	3.87	11.6	17.49	22.66	27.64

Table 20: q-Grams Multiple Pattern Search Algorithm Execution Time in Second for a file size 60 MB in Dhakshina Cluster-II with 10 nodes

		Number of patterns				
		1	5	10	20	50
Pattern length: m	m_1=5	0.67	1.98	2.87	3.78	4.55
	m_2=10	0.87	2.56	3.68	4.87	5.83
	m_3=15	0.82	2.66	3.92	5.15	6.26
	m_4=20	1.09	3.09	4.58	5.99	7.16
	m_5=25	1.11	3.27	4.79	6.37	7.69
	m_6=50	1.27	3.53	5.37	7.13	8.63
	m_7=100	1.59	4.48	6.66	8.87	10.65
	m_8=250	1.76	5.45	8.05	10.69	12.69
	m_9=500	1.89	5.89	8.63	11.58	14.09
	m_{10}= 750	2.19	6.42	9.39	12.69	15.23
	m_{11}=100	2.09	6.43	9.78	12.97	15.53
	m_{12}=200	2.32	7.11	10.17	13.72	16.57
	m_{13}=500	2.50	7.99	11.79	15.49	18.98
	m_{14}=100	2.68	8.16	12.18	15.77	19.19

6. ANALYSIS OF THE EXPERIMENTAL RESULTS

Aho-Corasick, is a trie of the patterns and search the text with the aid of the trie. The trie grows quite rapidly as the pattern set grows. For σ = 256, m = 8 and 100,000 patterns the trie takes 500 MB of memory. So trie-based algorithms are not practical for large pattern sets. The Aho-Corasick algorithm can give only one character throughput per transition or clock cycle, Searching using this algorithm will have a time complexity of O(n) where n is the amount of letters in the document. This algorithm allows searching for as many words of any length without an impact on searching performance. Another advantage is that once the DFA and failure function have been constructed, to use on any amount of different documents without having to reconstruct them. AC algorithm is a linear-time algorithm and is optimal in the worst case situations.

The running time of Wu-Manber has improved once the number of patterns exceeds about 7500. The reason for that is related more to the way greps work rather than to the specific algorithm. Agrep (and every other grep) outputs the lines that match the query. Once it is established that a line should be output, there is no need to search further in that line. Above 7500, the number of patterns becomes so large; most lines are matched and matched early on. So less work is needed to match the rest of the lines. We present this as an example of misleading performance measures; we probably would not have thought about this effect if the numbers had not actually gone down.

Advantage of AC-Bitmap algorithm is to decrease the size of memory. However, the performance of AC-Bitmap implemented by software is bad because of the extremely heavy cost of pop count and the search time of linked list. When AC-Bitmap algorithm is implemented by hardware, all data structures must be stored in wide embedded memory for performance issue, which makes high cost. [5]

q-Grams algorithms showed to be faster than Wu-Manber solutions for sets m_i=15, 25, 250, 500, 1000, 2000 and 5000. The result in q grams algorithm do not provide clear indication of better result. The result varies based on the filtering on text. Theory of q-Grams says that pattern of 1,000–10,000 and above patterns have a good performance. The gain is due to the improved filtering efficiency caused by q-Grams. It is expected that by increasing the efficiency of filtering mechanism by parallel algorithms, better performance can be achieved with q-Grams soon.

7. CONCLUSION

Parallel implementation of Aho-Corasick NFA and Wu-Manbar Multiple Pattern Searching algorithms, AC-Bitmap algorithm and q-Grams Search Algorithms were tested. The experiments recommend Wu-Manbar Multiple Pattern Searching algorithm for any length of alphabets and patterns, as the searching time is less compared to other method. The performance of AC-bitmap method is not as good as the implementation with linked list and without embedded memory. This implementation is just a study on popular existing algorithm on the present cluster computing infrastructure of Dakshina Cluster Series I & II. The result proves the order of searching for these algorithms. Speed up factor do not exactly matches with the result. It is because of the potential bottlenecks such as network communication overheads, memory bandwidth and I/O bandwidth[14]. Since size of the problem is fixed, Gustafson's Law is not applicable with this search results. Future work of this experiment is to find the better search results in another multiple string matching algorithms by modifying q-Grams and find the factors that affect searching time in a cluster computing environment.

8. ACKNOWLEDGEMENT

This work was supported fully by the Center for High performance Computing of Federal Institute of Science and Technology [FISAT]™, Angamaly, Cochin.

9. REFERENCES

[1] Sellers. P, The theory and computation of evolutionary distances: pattern recognition. Journal of Algorithms 1, 359–373, 1980.

[2] http://www.snort.org

[3] A. V. Aho and M. J. Corasick. "Efficient string matching: An aid to bibliographic search." Communications of the ACM, 18(6), 1975, pp.333–340.

[4] Sun Wu , Udi Manber, A fast algorithm for multi-pattern searching (1994)

[5] http://webglimpse.net/pubs/TR94-17.pdf

[6] N. Tuck, T. Sherwood, B. Calder, G. Varghese, "Deterministic memoryefficient string matching algorithms for intrusion detection," In Proceedings of the IEEE INFOCOM Conference, 2004, pp. 333–340.

[7] L. Salmela, J. Tarhio and J. Kytojoki: "Multi-pattern string matching with q-grams. ACM Journal of Experimental Algorithmics", Volume 11, 2006.

[8] Prasad J.C., K.S.M.Panicker, 'Single pattern search implementations in a cluster computing environment', IEEE Xplore Digital library, Digital Ecosystems and Technologies(DEST), 2010 4th IEEE International Conference 13-16 April 2010 on ISSN:2150-4938, Page 391-396.

[9] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary,[Sept 2008], HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, Innovative Computing Laboratory, University of Tenneesse. doi: http://www.netlib.org/benchmark/hpl/

[10] Prasad J.C., K.S.M.Panicker,[2009] 'Beowulf Dakshina Cluster Architecture with Linux Debian Operating system for MPI Programming', Proceedings of International Conference on Information Processing,, Bangalore, India ISBN: 978-93-80026-75-2, Page 350.

[11] P.D.Michailidis, K.G.Margaritis[2000], Parallel String Matching Algorithm: A bibliographical review, Technical Report, Dept.of Applied Informatics, University of Macedonia.

[12] Panagiotis D. Michailidis and Konstantinos G. Margaritis[2001], 'Parallel Text Searching Application on a Heterogeneous Cluster of Workstations', IEEE Computer Society Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'01), Page. 153

[13] Mansoor Alicherry, M. Muthuprasanna, Vijay Kumar, 'High Speed Pattern Matching for Network IDS/IPS', 2006 IEEE

[14] 'Replica Selection in the Globus Data Grid', A. V. Aho and M. J. Corasick. "Efficient string matching: An aid to bibliographic search." Communications of the ACM, 18(6), 1975, pp.333–340