An Approach to Assure QoS of Machine Translation System on Cloud

Pawan Kumar
Expert Software Consultants Ltd
New Delhi, India
hawahawai@gmail.com

Rashid Ahmad
LTRC, International Institute of Information Technology
Hyderabad, India
rashid.ahmed@research.iiit.ac.in

B. D. Chaudhary
Motilal Nehru National Institute of Technology
Allahabad, India
bdc@mnnit.ac.in

Mukul K Sinha
Expert Software Consultants Ltd
New Delhi, India
mukulksinha@gmail.com

Abstract*—Transfer based Machine Translation (MT) System is a large complex functional application. When these MT systems are deployed with increasing translation load the Quality of Service (QoS) degrades (namely, *job completion time* increases, *system throughput* decreases, and system performance does not scale with increase in provision of resources). To improve QoS of the MT system MapReduce framework for distributed processing was explored. MT application, which has very large code size (order of 100 MB) of computation, transferring it across the data nodes of the cluster would be totally antithetical to the basic goal of throughput enhancement. To utilize the benefit of parallelism provided by Hadoop, a very large complex MT application has adopted a distinct approach to overcome this difficulty with no time penalty. This paper presents an engineering approach to delude MapReduce framework for parallelization of machine translation tasks on a large cluster of machines to assure QoS of MT system. This paper reports the initial results of the experiments done in our laboratory by running MT System under cluster of virtual machines in private cloud. Further this paper asserts that, with the availability of *elastic* computing resources in cloud environment, the job *completion time* for any translation, irrespective of its size, can be assured to be within a *fixed time limit.

Keywords-Quality of Service; Machine Translation; Virtual Appliance; Natural Language Processing.

I. INTRODUCTION

Sampark is a machine translation (MT) system that applies transfer based approach to translate text documents among nine pairs of Indian languages [1]. Sampark system was deployed and released for public use at Sampark website for interactive as well as batch usage in 2008 [27]. The overview of this MT system comparing its transfer based approach (comprising three steps, viz., *analyze*, *transfer*, and *generate*) of machine translation to that of statistical based approach, followed by Google and Microsoft has been briefly reported in [2]. As the system was not designed a priori for scaling, its performance, with the increase in number of translation job requests, degrades sharply. Provisioning of additional computing resources, and employing load balancer, did not improve the overall system performance incrementally. With increase in number of jobs there is either degradation, or absence of improvement in the Quality of Service (QoS) of the system, mainly in three dimensions, viz.

a) Job *completion time* (solution time) increases fast
b) System *throughput* decreases (number of sentences translated per unit time) and
c) System performance (with provision of additional computing resources) does not scale linearly.

An MT system is a very complex application with a large code size of the order of 100 MB. It is a *functional* application where one sentence in the source language is translated into one sentence in the target language. To explain further, all the modules of a MT system produce same result given same input text, output does not depend on any *hidden information* or *state* as the program execution proceeds or between different executions of the program. An MT system treats its input text data as a list of sentences. Translation of each sentence is done independently, and has no effect either from its preceding or from following sentences. Further, it is also a compute intensive application as it takes quite a long time to translate a sentence. On an Intel(R) Core(TM)2 Quad CPU Q8300 @ 2.50GHz, L2 Cache 2048 KB, translating a sentence (average sentence size 10 words) takes approximately 3 seconds. As the compute cost is the product of number of compute resources and its utilization time to execute a job, the compute cost to translate a single sentence is 3 seconds.

An MT system like Sampark that translates a text document from a source language to a target language may have jobs that have large variance in their input data size (*workload*). On one end there may be a job to translate a single sentence, to other translating a newspaper of 30 pages, or yet another job translating a book of 500 pages. In spite of provisioning of additional computing resources, the completion time of a large job cannot be reduced. A large job does not get advantage of available and unused computing resources as a load balancer assigns each job, irrespective of size of its workload, to a distinct computing resource. This limitation caused due to the specific nature of MT

 ISBN: 978-1-61208-271-4

application forced us to explore the applicability of MapReduce [3] parallelization framework to reduce the completion time of large machine translation jobs.

The Map Reduce framework is suitable for functional applications as it is able to split a large job into multiple job partitions, and each job partition can run on different computing nodes in parallel. This approach of parallel execution of job partitions not only reduces a job's completion time, it also facilitates the better utilization of available computing resources. The MapReduce programming model has been designed for applications that expect provisioning of on-demand service model for computing resources. The Cloud computing platform comprising large clusters of machines provides, on-demand, availability of computing resources of desired size and number that can be scaled up/down incrementally [4].

This paper presents an engineering approach, utilizing Hadoop [5], the open source implementation of Map Reduce framework, to partition each large MT job into multiple job partitions, to run them, in parallel, on a given cluster of virtual machines provided by private Eucalyptus Cloud [6] set up in our laboratory. This parallel execution of the job partitions reduces the job completion time, and also enhances utilization of the given compute resources.

In the cloud computing environment, in addition to the reduction in job completion time, there is need to enhance the system throughput, as well. Then only it ensures the best utilization of computing resources, resulting in increase in the overall system performance, giving us the cost benefit of cloud computing environment.

The set of three experiments that we conducted show that:

a) for a large job of any size, the job completion time can be reduced with increase in computing resources,

b) there is an optimum job partition size (described in Section V, Experiment Two) that ensures nearly the best system throughput (i.e., number of sentences translated per unit time), and

c) the optimum job partition size also ensures best utilization of available computing resources, resulting in completion of each job with least computation cost, assuring, in turn, very high overall system performance.

In this way, our approach assures all the three dimensions of the QoS of the MT system. MT system is an example of class of Natural Language Processing (NLP) applications that are functional in nature. This engineering approach to assure QoS can be applied to other similar applications like, text-to-speech, speech-recognition, and text-summarization, etc.

In Section II, an overview of the Map Reduce Framework is given, including its strengths and limitations while the Section III lists related works, its adaptation for various types of applications, and also for various types of platforms. In Section IV, our approach to employ Map Reduce techniques is discussed that assures the QoS for the Sampark MT system, and Section V gives the details of our experimental results. And finally, Section VI presents our conclusion.

II. HADOOP MAP-REDUCE FRAMEWORK: OVERVIEW, STRENGTHS AND LIMITATIONS

A. MapReduce: An Overview

MapReduce has become the most used parallelization framework in the data centers comprising of commodity computers [7]. MapReduce is mostly suited for functional applications, and its two functions that is *map* and *reduce* are inspired from LISP, the *functional programming* language [8].

The Hadoop Framework, the open source variant of Map Reduce, is composed of Hadoop MapReduce, and Hadoop Distributed File System (HDFS). HDFS is used to store both input data to the *map* step and the output data from the *reduce* step. A Hadoop installation is comprised of a *cluster of nodes*, consisting of a *master node*, called the *JobTracker*, and several *worker nodes*. The *JobTracker* is responsible for accepting the jobs from the clients, and splitting each job into multiple job partitions, and assigning those job partitions to be executed by different worker nodes. Each worker node runs a *TaskTracker* that executes currently assigned task to it, and on its completion, informs the same to the *JobTracker*. By communicating with each *TaskTracker*, the *JobTracker* keeps track of all the running job partitions, and also schedules of new job partitions to worker nodes that are free.

In Hadoop, the input data of a job gets distributed on the worker nodes of the cluster while it is being loaded. The Hadoop Distributed File System (HDFS) splits the input data into chunks, and each chunk is loaded on different nodes of the cluster, well before the application gets initiated.

When the JobTracker assigns a job partition to a worker node it sends the program code to that node. *It is presumed that the time spent in transferring the program code to the worker node is relatively very small in comparison to the execution time of the job partition.*

B. Strengths

The main advantage of MapReduce programming model is its simplicity. The user has to specify his algorithm as a pair of *map* and *reduce* tasks that conform to the programming model. *A functional application whose input data can be represented as a list can always be modeled in MapReduce framework*. The rest of the details, like, workload partitioning, distributed execution, network communication, coordination, and fault tolerance, etc., are all handled by the MapReduce framework itself.

This model of Map-Reduce is very efficient primarily for batch jobs, and also for those functional applications that have relatively smaller code sizes and operates on extremely large input data sizes.

C. Limitations

The intrinsic limitation of MapReduce is its *one-way scalability of its design*, i.e., to scale up to process very large data sets [9]. Again, it handles large data sets that are at rest, but is unable to handle large data in motion that can come as stream [10].

 ISBN: 978-1-61208-271-4

In the present implementation of MapReduce in Hadoop, the program code gets transmitted across the worker nodes of the cluster. And hence, for an application that has very large code size transferring it across the worker nodes would completely drain its job completion time enhancement due to parallel processing of its job partitions. Thus, the main limitation of Hadoop MapReduce is that it is completely unsuitable for jobs with large code size.

To utilize the benefit of parallelism provided by Hadoop, a functional application with large code size is required to evolve a distinct approach to overcome this difficulty with no transfer time penalty.

III. RELATED WORK

The MapReduce framework that was originally proposed by Google is being utilized by it to process more than 10 petabytes of data per day [3]. After the release of Hadoop implementation of the MapReduce framework more than hundred organizations, including large companies and academia are using it for various types of applications. This has also resulted intense research and development activities in various directions [11]. Some researchers have developed of many distinct MapReduce algorithms for processing of different types of massive data [12, 13], some have simulated well known parallel processing algorithm in MapReduce framework [14], while some others are involved in developing schemes for implementing MapReduce framework in distinct types of physical platforms [15, 16], and in optimizing the scheduling problem in its context [17].

The quality of output of Statistical Machine Translation (SMT) Systems increases with the increase in amount of their training data [18, 19]. Good SMT systems usually train their translation engines on 5-10 million sentences pair corpora, and to train engine on such massive volume of data, even on good processing platforms, takes couple of days to even a week. And hence, many efforts are being pursued to use MapReduce framework to execute such training module over large corpora on a large distributed systems, bringing down the training time within couple of hours [20]. Hadoop MapReduce framework has been used to study throughput improvement of SMT system [18, 19, 20, 21]. Open source toolkits capable of training phrase based SMT models on Hadoop cluster [22] and grammar based SMT on Hadoop cluster [23] have been reported.

IV. TO ASSURE QOS OF SAMPARK MT SYSTEM: AN ENGINEERING APPROACH

First, we have tried to abstract those distinguishing features of our application, viz., the transfer based MT system Sampark, that makes it an attractive application for MapReduce framework, and they are:

- A transfer based MT system is a functional application, and hence, MapReduce framework would be applicable to it,
- Any text document file that is required to be translated, i.e., data input to the MT system, can always be abstracted as a list of paragraphs, or a set of sentences of any required size, and hence, it can be easily parallelized and executed on large cluster of machines [24],
- The incremental scaling up of computing resources on-demand is integral part of any MapReduce framework, whether it is a cluster of multi-core physical machines, or large set of virtual machines in the cloud [4]. And hence, we would be able to assure all the three dimensions of QoS (discussed in the Section I: Introduction) of MT system.

A. Hurdle: To Run Application with Large Code Size on Hadoop

The Hadoop uses strategy of *moving computation to the data site*, instead of moving the data to the computation site. This strategy allows Hadoop to achieve *high data locality* which, in turn, results in high performance.

As discussed earlier, the Sampark MT system is a very large and complex application with large code size of approx. 220 MB. This code comprises of around 100,000+ lines of code (in various programming languages), including the lexical resources, the rule base, and the machine learned data, each is of very large size, required by its various modules to perform their functionality. Transferring such a large code to each worker node would create large communication load draining completely the advantages achieved by parallel processing of job partitions.

B. Solution: Sampark MT System as a Virtual Appliance

To circumvent the above problem of transferring large code size to each worker node, the Sampark MT system is packaged as Virtual Appliance [25]. An MT virtual appliance is a full application stack containing the Just enough Operating System (JeOS), the Sampark MT system, the Hadoop system, their required dependencies, and the configuration and data files required to run the MT system. Everything is pre-integrated, pre-installed, and pre-configured to run on a virtual machine.

Whenever a new VM is provisioned from cloud, an image of the Sampark virtual appliance is actually instantiated on the new VM. For a dedicated application environment, this engineering approach completely avoids the need of transferring the MT computation code to worker nodes at run time. This technique facilitates new nodes to be added on demand.

C. Implementation: To run Sampark MT System with Large Code Size under Hadoop

To circumvent this problem for running MT System on a Hadoop, we have taken following three steps:

- We have developed a program, called *mtclient* that runs on the Hadoop *master node*. Traditional implementation of MapReduce expects data to be partitioned well before the MapReduce job is executed. This *mtclient* partitions the workload and submits the job for translation to the Hadoop master
- *mtmap* is another program that is invoked by Hadoop master for each of the workload partition. The code of *mtmap* is transported to each worker node for execution of the map tasks.

 ISBN: 978-1-61208-271-4

- *mtmap* in turn calls *mtmain*, which is part of the Sampark virtual appliance. *mtmain* is the main translation system that takes list of sentences as input and produces a list of sentences as translated output.

Once all the *map* tasks are over, Hadoop master calls *mtreduce* to collate the output translation. In this way, we have deluded Hadoop to run a large machine translation job as set of parallel map tasks in a dedicated application scenario.

V. THE EXPERIMENTAL SETUP, SET OF EXPERIMENTS, AND THER RESULTS

The experiment has been done on Hindi to Punjabi Sampark MT system to measure the various QoS dimensions of the system. The Sampark MT system (program codes along with lexical resources, rule bases, and machine learned data) is packaged as a virtual appliance [25]. The Sampark MT virtual appliance that we used for performing our experiments was based on CentOS-5.7 as host OS, with Xen as virtualization layer, along with Hadoop 0.20.2 as middleware for work load partitioning.

All the experiments are performed on similar virtual machines in the Eucalyptus cloud. Each of the virtual machines in the cloud are 2 CPUs, 1GB RAM with CentOS-5.3 (64-bit) as guest OS. For our experiments we had allocated 10 worker nodes in the cloud. On each worker node, the Sampark MT virtual appliance was pre-installed as a part of the setup.

We conducted three different types of experiments with different number of compute resources, and different data sets as it was required by the experiments (for experiment one 1500 sentences, for experiment two 3000 sentences, and for experiment three the data set varies from 200 to 25600 sentences). As the virtual compute resources are homogeneous in nature, and to make the data sets homogeneous in nature, we have replicated a set of 10 sentences (with average size of 8.5 words) repeatedly, to get the required size of experimental data sets.

A. *Experiment One: To Investigate the Relation of Job Completion Time with respect to the Amount of Compute Resources*

In this set of experiments, each experiment was done, for a given number of virtual nodes in the cloud, and with the fixed job size of 1500 sentences with increasing number of job partitions (also called *task*). The job partition sizes used in experiments are 5, 10, 15, 20, 25, 50, 75, 100, and 150 sentences each.

The same experiment was repeated with increasing the number of virtual nodes in the cloud, viz., node clusters of 2, 4, 8, and 10.

The same experiment was earlier performed on a standalone system with same virtual machine configuration in the cloud but without Hadoop.

When we have small job partition size, for a given job the number of job partitions would be large. And hence, for a given number of virtual nodes, to run all job partitions (to complete the job), it would take multiple cycles of run. *In comparison to a job partition (task) execution time, the inter-cycle run overhead would be negligible.*

Table I shows the job completion time with increasing number of virtual nodes, and with increasing size of job partition. From this set of experiments we conclude:

- For a given job, the job completion time reduces with the increase in computing resources,
- The reduction in job completion time is linear in the beginning, but starts saturating beyond a certain point

TABLE I. SHOWING JOB COMPLETION TIME IN SECONDS FOR 1500 SENTENCES

Partition Size (Sentences per Task)	Job Completion Time (in Seconds)				
	10 Nodes	8 Nodes	4 Nodes	2 Nodes	1 Nodes*
5	258	302	583	1150	2704
10	173	215	402	798	1979
15	139	176	312	631	1704
20	137	167	285	566	1803
25	130	171	305	528	1487
50	119	134	284	433	1275
75	152	104	174	363	1193
100	151	119	211	362	1412
150	152	194	397	324	1385

* This experiment was done on a single virtual machine without Hadoop

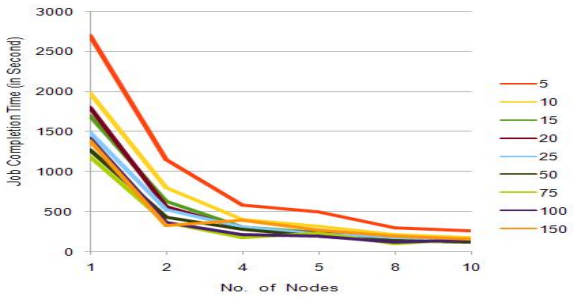

Figure 1. Job Completion time vs. No. of Nodes

B. *Experiment Two: To Investigate the Relation of Job Partition Size with respect to Throughput.*

In this set of experiments, we increased the size of data set to 3000 sentences, mainly to reduce the influence of inter-cycle run overhead on the throughput. Larger is the job completion time lesser would be the influence of inter-cycle run overhead. Each set of experiment the job partition sizes used were 5, 10, 15, 20, 25, 50, 75, 100, and 150 sentences each. This variation in job partition size is the same as in Experiment One.

Again, to focus our attention on throughput we have conducted only two sets of experiments on two compute resource configurations, viz., 5 and 10 virtual nodes.

Again, to focus our attention on throughput we have conducted only two sets of experiments on two compute resource configurations, viz., 5 and 10 virtual nodes.

Table II enumerates the results of the two sets of experiments. The result shows that, for a given job the best

throughput is achieved at a particular job partition size, irrespective of number of compute resources utilized. By increasing job partition size, the improvement in throughput is not very significant. *As we have reached the rim of the best throughput, we call this job partition size as the optimum job partition size.*

TABLE II. SHOWS COMPUTATION COST VS PARTITION SIZE FOR 3000 SENTENCES

No of Tasks	Partition Size (Sentences per Task)	10 Nodes	5 Nodes
600	5	35	37
300	10	49	53
200	15	59	64
150	20	63	69
120	25	68	72
60	50	86	90
40	75	93	97
30	100	82	104
20	150	101	107

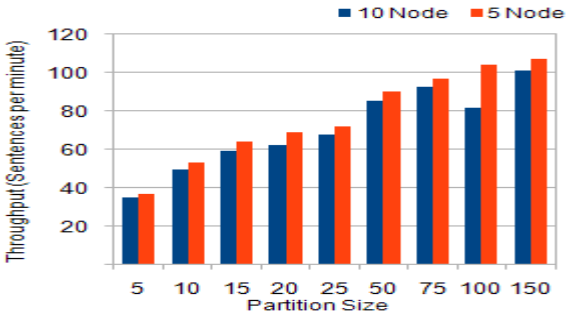

Figure 2. Throughput vs partition size of task in term of number of sentences

C. *Experiment Three: To Investigate the Relation between Job Partition Size and Throughput.*

In this case, we have conducted 3 sets of experiments, each with the same compute resource configuration of 5 virtual nodes.

As we are varying the job partition size to observe that where the throughput is the maximum, in each set of experiment we have maintained a fixed number of job partitions (tasks). To keep fixed number of partitions while varying the job partition size, we have to increase the job size i.e., number of sentences. The 3 sets of experiments have 40 tasks, 60 tasks and 80 tasks respectively. Figure 3 shows throughput verses partition size of task.

Table III enumerates the results of the three sets of experiments done. These results show that for a given job the best throughput is achieved at a particular job partition size. It also shows that by changing the job size (i.e., the number of sentences) hardly changes the optimum job partition size. Increasing the partition size beyond the optimum job partition size does not enhance the throughput significantly. We see that, in this range, if the partition size is doubled, the throughput increases by less than 5%.

TABLE III. SHOWS TIME TO TRANSLATE A GIVEN TASK FOR VARIOUS PARTITION SIZES ON A 5 NODE CLUSTER FOR 25600 SENTENCES

No of Tasks	Partition Size	Total Sentences	Total Compute Time in seconds	Throughput per minute
80	10	800	1055	45
80	20	1600	1475	65
80	40	3200	2340	82
80	50	4000	2620	92
80	80	6400	3750	102
80	100	8000	4455	108
80	160	12800	6665	115
80	200	16000	8240	117
80	320	25600	12920	119
60	10	600	800	45
60	20	1200	1025	70
60	40	2400	1715	84
60	50	3000	2005	90
60	80	4800	2830	102
60	100	6000	3320	108
60	160	9600	5055	114
60	200	12000	6140	117
60	320	19200	9990	115
40	10	400	570	42
40	20	800	855	56
40	40	1600	1165	82
40	50	2000	1355	89
40	80	3200	2115	91
40	100	4000	2275	105
40	160	6400	3435	112
40	200	8000	4175	115
40	320	12800	6430	119

TABLE IV. SHOWS THROUGHPUT VARIATIONS FOR VARIOUS PARTITION SIZES

Partition Size (Sentences per Task)	80 Task	60 Task	40 Task
10	45	45	42
20	65	70	56
40	82	84	82
50	92	90	89
80	102	102	91
100	108	108	105
160	115	114	112
200	117	117	115
320	119	115	119

VI. CONCLUSION AND FUTURE TASKS VISUALIZED

This paper presents the engineering approach that we have developed to run a functional application like MT system with a large code size as a dedicated application in MapReduce Framework, to get enhanced QoS utilizing its list homomorphism characteristics [24] for parallel

execution. This approach to assure QoS can be applied to a large group of NLP applications.

We have also developed a scheme to delude Hadoop MapReduce framework to load the MT system with large code size (by packaging MT as a virtual appliance), a priori on all worker nodes, to overcome the transfer cost at run time.

Contribution of our work is threefold:

- Completion time for any large job can be reduced with increase in computing resources,
- There exists an optimum size of job partition for which the best system throughput is achieved,
- The minimum completion time along with the best system throughput would incur the minimum compute cost in the cloud environment.

In this way, our approach assures all the three dimensions of the QoS of the MT system. In future we plan to extend this approach to other NLP applications that exhibit list homomorphism and can be partitioned for distributed computing.

ACKNOWLEDGMENT

We would like to thank Prof. Rajeev Sangal for his help to setup the experiments in their laboratory.

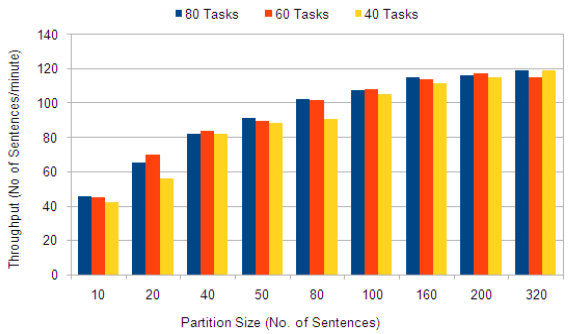

Figure 3. Throughput vs. partition size of task

REFERENCES

[1] R. Sangal, “Project proposal to develop Indian language to Indian language Machine Translation System”, IIIT Hyderabad, TDIL Group, Dept. of IT, Govt. of India, (2006).

[2] G. Anthes, “Automated Translation of Indian Languages,” CACM, vol. 53 (1), Jan. 2010, pp. 24-26, doi: 10.1145/1629175.1629184.

[3] J. Dean and S. Ghemawat, “MapReduce: A Flexible Data Processing Tool,” In Proc. of OSDI'04: Sixth Symp. on Operating System Design and Implementation, pp. 137-149, Dec. 2004.

[4] M. Armbrust, A. Fox, R. Griffith, “Above the Clouds: A Berkeley View on Cloud Computing,” Technical Report No. UCB/EECS-2009-28, Univ. of California, Berkeley, Feb. 10, 2009.

[5] The Apache Software Foundation, “Hadoop: MapReduce Framework” http://hadoop.apache.org [retrieved: February, 2013]

[6] D. Nurmi, et al., “The Eucalyptus Open Source Cloud Computing System,” In Proc. of 9th IEEE/ACM Intl. Symp. on Cluster Computing and Grid”, pp. 124-131, 2009.

[7] L. Barroso, J. Dean, and U. Hoelzle, “Web search for a planet: The Google cluster architecture,” IEEE Micro, vol. 23, no. 2, pp. 22-28, 2003.

[8] R. S. Bird, “An Introduction to the Theory of Lists,” Oxford University Technical Monograph PRG-S6, 1986.

[9] Z. Ma and L. Gu, "The Limitation of MapReduce: A Probing Case and a Lightweight Solution," In Proc. of the 1st Intl. Conf. on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2010). Nov. 21-26, 2010.

[10] V. Kumar, H. Andrade, Bugra Gedik, and K. Lung Wu, “DEDUCE: At the Intersection of MapReduce and Stream Processing,” In Proc. of the 13th Intl. Conf. on Extending Database Technology, pp. 657-662

[11] J. Lin and C. Dyer, “Data-Intensive Text Processing with MapReduce,” University of Maryland, USA, April 2010.

[12] B. He, W. Fang, Q. Luo, N.K. Govindarajan, and T. Wang “Mars: A MapReduce Frameworks on Graphics Processing,” In Proc. of 17th Conf. on Parallel Architecture & Compilation Techniques, pp. 260-268, 2008.

[13] M.de Kruijf and K. Sankarlingam, “MapReduce for the Cell B.E. Architecture”, University of Wisconsin, Comp. Sc., Tech. Report: CS-TR-2007-1625, 2007.

[14] H. Karloff, S. Suri, and S. Varrilvitski, “A Model of Computation for MapReduce,” In Proc. of 21st Annual ACM-SIAM Symp. on Discrete Algorithm, 2010.

[15] G. Ananthnarayanan et al., “Reining in the Outliers in Map-Reduce Cluster using Manti,” In Proc. of USENIX OSDI, 2010.

[16] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, “Evaluating MapReduce for Multi-core & Multi-processor systems,” In Proc. of the IEEE 13th Intl. Sym. on High Performance Computer Architecture, pp. 13-24. Phoenix, Arizona, 2007.

[17] H. Chang et al., “Scheduling in MapReduce like System for Fast Completion Time”, Bell Labs Alcatel-Lucent, Purdue University, 2011.

[18] M. Banko and E Brill, “Scaling very very Large Corpora for Natural Language Disambiguation,” In Proc. of 39th Annual Meeting of Assoc. of Computational Linguistics (ACL 2001), pp. 26-33, Toulouse, France, 2001.

[19] C. Collism-Burch, C. Bannard, and J Schroeder, “Scaling Phrase-based Statistical Translation to Larger Corpora and Larger Phrases,” In Proc. 43rd Annual Meeting of Assoc. of Computational Linguistics ACL, pp. 255-262, Ann Arbor, Michigan, USA, 2005.

[20] C. Dyer, A. Cordora, A. Mont, and J. Lin, “Fast, Easy & Cheap: Construction of Statistical Machine Translation Model with MapReduce,” In Proc. of 3rd Workshop on Statistical MT at ACL, University of Marytal, Columns, Ohio, 2008.

[21] R.M.Yoo, A. Romano, and C. Kozyrakis, “Phoenix Rebirth: Scalable MapReduce on a Large Scale Shared Memory System,” Stanford University, Computer System Laboratory, CA, USA, 2009.

[22] Q. Goa and S. Vogel, “Training Phrase-based Machine Translation Models on the Clouds: Open Source Machine Translation Toolkit Chanki,” The Prague Bulletin of Mathematical Linguistics, 93: pp. 37-16, 2010.

[23] V. Ashish and A. Zollnam, “Grammar Based Statistical MT on Hadoop.An end-to-end Toolkit for Large Scale PSCFG based MT”, The Prague Bulletin of Mathematical Linguistics (91), pp. 67-78, 2009.

[24] M.Cole, “Parallel Programming with List Homomaphism”, Parallel Processing Letters vol. 5, No. 2, pp. 191-203, 1995.

[25] P. Kumar, R. Ahmad, B. D. Chaudhary, R. Sangal, “Machine Translation System as a Virtual Appliance: For Scalable Service Deployment on Cloud,” In Proc. of IEEE 7th Intl. Symp. on Service Oriented System Engineering (SOSE 2013), pp. 304-308, 2013.

[26] R. Ahmad, et al., “Enhancing Throughput of a Machine Translation System using MapReduce Framework: An Engineering Approach,” In Proc. of 9th International Conference on Natural Language Processing (ICON-2011), pp. 200-206, 2011.

[27] Sampark: Machine translation system among Indian languages, http://sampark.org.in [retrieved: January, 2013]

 ISBN: 978-1-61208-271-4