IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-8 E-ISSN: 2347-2693

Design of High Performance, Scalable Content-based Publish-Subscribe System using MPI-CUDA Approach

M.A. Shah[1*], D.B. Kulkarni[2]

[1*]Dept. of CSE, Walchand College of Engineering, Sangli (M.S), India
[2]Dept. of CSE, Walchand College of Engineering, Sangli (M.S), India

[*]*Corresponding Author: medha.shah@walchandsangli.ac.in Tel.: +919423872296*

Available online at: www.ijcseonline.org

Received: 12/Jul/2017, Revised: 23/Jul/2017, Accepted: 16/Aug/2017, Published: 30/Aug/2017

***Abstract*—** *Today* Publish-subscribe model is used as communication backbone for various application domains such as IoT, Social networking, Intrusion detection system and Financial trading. Content-based flavor of Pub-Sub system enables routing of information from producers to consumers based on contents of the query or depends on subscriptions entered by the user. In this model, information is disseminated from producers to consumers through a network of brokers. The significant challenge in content-based Pub-Sub system lies in an efficient matching of an event against a large number of subscribers on a single message broker. To provide high throughput service guarantee to the subscriber of Pub-Sub system we propose a novel hybrid model for parallel event processing using MPI-CUDA approach. This approach combines message passing interface (MPI) and CUDA, a parallel computing platform and programming model, invented by NVIDIA. Results are compared with CCM (Cuda Content Matching Algorithm), a high performance, and parallel content matching algorithm. Approximately 1.77X speedup is observed in matching latency. This approach is suitable for use in event processing of large data intensive applications where the rate of arrival of the event is high.

***Keywords*—** Matching Latency, MPI-CUDA, High performance, parallel event processing.

I. INTRODUCTION

Events are everywhere. New sources of events like social feeds, IoT devices, RFID tags, cameras, mobile devices, internet services, web sites and data repositories generate events at an enormous rate. The amount of data is growing exponentially. Every day at least 2.5 quintillion bytes of data get produced. The growth in the size of data is exponential.
Many applications want to exploit these events in real time. Many distributed applications use Pub-Sub communication paradigm as communication backbone. In Pub-Sub system, Subscribers receive total or subset of the total messages published by one or more publisher. Here receivers declare their interest in the particular event in the form of subscription. The publisher publishes the information of interest as message or notification. Content-based Pub-Sub delivers published messages to the interested subscribers through event notification system. In content-based Pub-Sub system, event matching plays a critical role. Matching of an event against a large number of subscribers is carried out on a single message broker. To minimize matching latency and to deliver high throughput are the two fundamental goals of the Pub-Sub system. As various parallel architectures and frameworks are easily available at a reduced cost now, parallel and scalable content-based matching algorithms are to be designed and deployed to achieve high throughput.

In this paper, we present research contributions. There are recent studies on the development of a high-performance content-based system using GPU [1]. As an example, a Cuda content matching algorithm proposed in [1] achieves low matching latency and provides a speedup of 148.7 compared with SFF, a sequential counting algorithm. Previous work [1] commonly assumes that all subscriptions can fit into the memory of single GPU. But the performance of existing matching algorithms degrades for a lot of subscriptions and events. But best of our knowledge, 32-64 GB is a typical size for the system memory, whereas a single GPGPU has between 4 and 12 GB device memory. When dealing with data-intensive applications, the size of the device memory may thus become a limiting factor. As all current generation machines equipped with multiple GPU cards, we address this memory problem by using Multi-GPU and MPI-CUDA approach. Workload gets distributed among available GPUs and hence it also solves the problem of overloading of broker for large data sets of subscriptions. Here we only present the

MPI-CUDA approach of parallel event processing.

This paper is organized as follows. Section II describes some related work in the area of traditional as well as the high-performance content-based Pub-Sub system. Section III presents data model typically assumed for same. Section IV presents the MPI-CUDA approach of developing high-performance Pub-Sub system. Section V shows experimental setup and results of the proposed approach.

II. LITERATURE SURVEY

We first review related work on traditional distributed content based Pub-Sub systems, followed by recent work in the area of high-performance Pub-Sub systems.

A. General Pub-Sub Research

Earlier work related to Pub-Sub system has relied on networks of brokers also called dispatchers, which are dedicated machines, arranged in hierarchical or tree based topology, perform the various operations including (1) Subscription management for users and other brokers (2) matching of incoming publications against stored subscriptions (3) delivering notifications to subscribers. Several Pub-Sub systems (e.g., SIENA [2], Gryphon [3], PADRES [4], HERMES [5]) proposed earlier follow a content-based addressing scheme for subscriptions. All these systems make use of an application-level network of brokers that does the task of matching.

B. Event processing algorithms in Pub -Sub Systems

Matching algorithms are categorized into counting-based [6, 7, 8] and tree-based [9, 10, 11] algorithms. These algorithms are further classified as key based and non-key based. The work proposed in [12] is key-based, where for every expression a set of predicates are selected as an identifier. Non-key based approaches are discussed in [6, 10, 8]. In counting based approach, an aim is to minimize predicate evaluations by constructing inverted index over unique predicates results in rigid clustering. In [7] propagation, a key based method was proposed while [8] discussed k-index, a non-key based method. Likewise, tree-based methods are designed to reduce predicate evaluations and to recursively divide the search space by eliminating subscriptions on encountering unsatisfied predicates. The most prominent tree-based method, Gryphon, is a static, non-key based algorithm [9]. BE-Tree [14, 15] is a novel tree-based approach, which also employs keys, which outperform existing work [6, 9, 7, and 8]. The latest improvement of counting based algorithms is k- index [8]. This algorithm scales well for thousands of dimensions and supports equality predicates as well as non-equality predicates. K-index is static and does not support dynamic insertion and deletion. BE-Tree is distinguished from k-index in many aspects. BE-Tree is dynamic and encourages richer predicate operators (e.g. range operators), and adjusts to workload changes. However, BE-Tree poses some limitations on attribute values. These values should be discrete in nature and their range is to be pre-specified. Additionally, BE-tree [14, 15] uses a clustering policy that becomes ineffective in certain cases. PUBSUB [13], which is a heterogeneous system, enable the users to select data structure best suited for each attribute by keeping track of the buckets in an attribute structure. Because of PUBSUB's heterogeneity in data structures for each attribute, PUBSUB [13] permits all attribute data types.

C. High-performance Pub-Sub systems

The idea of parallel matching has been recently addressed in a few research papers. In [16], the authors exploit multi core CPUs both to speed up the processing of a single event and to parallelize the processing of different events using threads. However, the processing delays and throughput reported seems to be much worse than those obtained by our Multi-GPU approach, due to the use of limited threads available for processing. Parallelization of the matching process using ad-hoc (FPGA) hardware is presented in [17]. The author in [1] described a new Pub-Sub content-based matching algorithm designed to run efficiently both on multi-core CPUs and CUDA GPGPUs. The algorithm takes a substantial amount of time for processing large numbers of subscriptions, filters, interfaces. The efforts are taken to deploy the Pub-Sub system on storm [18] architecture for fast matching using local as well as a distributed cluster. Resource utilization should be appropriate for better performance. Storm Pub-Sub system approximately produces 2200 event/s on a distributed cluster. As compared to our Multi-GPU approach this throughput is low. Stream Hub [19] is a novel Pub-Sub deployed on a cluster of 384 cores. This Pub-Sub is able to register and filter a large number of publications against stored subscriptions resulting nearly 400 K notifications/s. Deadline aware algorithm [20] is designed to maintain Quality of Service in Pub-Sub is modified, originally mentioned in [21]. In modified smart dispatch algorithm, the number of failures decreases with the increase in a number of cores. This paper uses the approach of parallelism using multithreading, so limited scalability is achieved. Our aim is to make high performance and scalable Pub-Sub system by processing events in parallel.

III. DATA MODEL

Here we illustrate what we mean by the event, filter, and predicate. An event is defined as a set of typed attributes. Each attribute in an event has a name and has a type and a value. For example, string class=travel/airlines/offer; date starts=Jun; date expires= Aug; string origin=LA; string destination= AUS; string carrier=United is an event. The event is also defined as an attribute value pair. A Filter is a conjunction of attribute constraints. Each attribute constraint has a name, a type, an operator, and a value. A constraint defines an elementary condition over an event. For example string class >*travel/airlines; date starts<Jul; date expires>Jul; string origin=LA; string destination= AUS is a valid filter matching the event of the previous example.

So a filter matches an event if all the attribute constraints in a filter are satisfied by the attributes in an event. A predicate is defined as a disjunction of filters. A Predicate matches an event if, at least one of its filter matches an event.

IV. MPI-CUDA APPROACH

This algorithm harnesses the power of MPI and CUDA distributed parallel programming model. MPI is a standardized API for communication between different processes. Generally, CUDA programming is used for parallel computing on a single computer or node. The basic idea of the distributed parallel programming model is to use MPI+CUDA to realize two-level parallel computing. MPI programming model helps to achieve a coarse-grained parallel computing between the computational nodes of the cluster. CUDA programming model helps to achieve a GPU-accelerated fine-grained parallel computing in each computational node. Here our aim is to scale content-based Pub-Sub system on multi node cluster and to achieve high throughput. The Structure of MPI+CUDA Programming is shown in Figure 1.

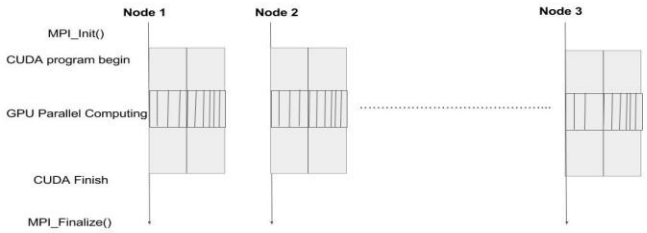

Figure 1. The MPI + CUDA Based Parallel Computing Program Model

A. MPI-CUDA Framework.

Here we propose two level task hierarchies. The proposed algorithm will use MPI for distributing subscriptions and events among available processes in the cluster and utilize the GPU for running a content matching algorithm. Figure 2. depicts the event dispatcher framework that processes events in parallel. The master process gets notified about a number of worker nodes available in the cluster. The master process sends the complete set of subscriptions to each process for matching. Assignment of a process to GPU is one to one. The master process sends the individual event (i.e. separate event for each process) to slave process. Slave process runs the matching algorithms on GPU and sends back the matching results to master process. The framework does the task of load balancing. As individual worker process sends the results to master process, immediately it receives a new event for processing. In this approach, we used non-blocking MPI subroutines for load balancing. Pseudocode of MPI-CUDA algorithm is presented in section C.

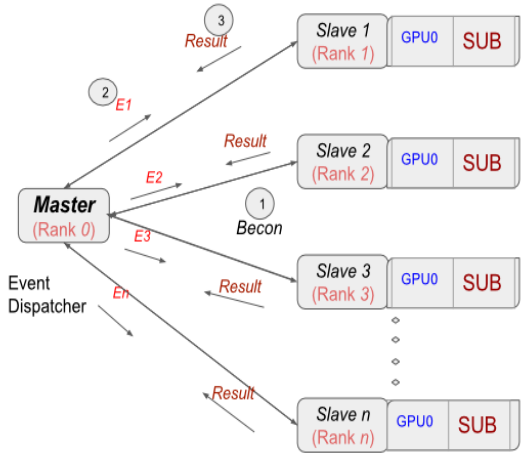

Figure 2. Event Dispatcher Framework

This approach uses CCM presented in [1] as a base algorithm. In the following section, CCM algorithm is explained in brief.

B. CCM Algorithm

CCM is parallel content matching algorithm designed to run on GPU. This is based on counting algorithm [22], prominently used for matching in Pub-Sub systems. The algorithm has three phases: a filter selection phase, a constraint selection phase, and a constraint evaluation and counting phase [1]. In the first phase, the set of filters is partitioned based on their attributes' names. In the second phase, for each attribute 'a', in the event, 'e', the set of constraints (part of the filters) having the same name as 'a' is

selected. Evaluation of selected constraints is carried out in the third phase using the value of each attribute 'a'. For each satisfied constraint 'c', the counter associated with filter 'f' is increased. When all the constraints of the filter are satisfied, a filter matches an even 'e' and so satisfies the predicate 'p' to which it belongs. When this process is completed, an event can be forwarded to the interface exposing predicate p. CCM algorithm evaluates multiple constraints in the filters simultaneously, by using GPGPU cores. The algorithm maintains following data structures for storage of events and subscriptions. Here onwards we refer CPU as host process. Host process maps each distinct attribute name to a single bit of small bit vector called NameVector (NVe). It also creates Filters and Constraints tables, to organize the constraints of the filters into five data structures namely ConstrOp, ConstrVal, ConstrFilterId, ConstrBF, and NumConstr [1], according to constraint name. Host process builds data structures named Filters and Interface [1] to maintain information about filter-size, filter-count, and interface-Id to which the filter belongs. An Interface is an array, indicates matched interfaces by setting the corresponding index of the array to one. For each event, host process builds the table Input. It includes one row for each attribute in event e. For each attribute 'a' in event, 'e', each row of the input table stores the value of 'a', its type, the number of constraints having the same name as 'a', and the pointers (in the GPGPU memory) to the rows of Filters and Constraint table that are relevant for a. All these data structures are transferred to the GPGPU for processing an event. CCM launches a kernel named evalConstraint, which uses thousands of GPGPU threads to evaluate constraints in parallel. Each thread evaluates a single attribute 'a' of e against a single constraint c. The results of the computation (i.e., the Interfaces array) are copied back to the host memory and the Filters Count and Interfaces structures are reset for processing of the next event.

C. Algorithm 1: MPI-CUDA

For master process:

Transfer Filters and Constraints data to all processes.
  While all events are not processed
    Receive matched interfaces from `x' process
    Send an incoming event to `x' process.
  End While
For slave process:
  Receive Filters and Constraints data from master process
    Copy received data to memory of every GPGPU asynchronously
  While (True)
    Send a dummy array with no interfaces matched.
    Receive event from master process
      If number of attributes in event equals zero
        Break
      Else
        Process event by calling kernel of GPU evalConstraint<<<NUM_BLOCKS_NUM_THREADS>>>
        Send matched interfaces to master process
      End If End while.

V. EVALUATION

Here we want to compare our work with state-of-the-art [1] CUDA content matching algorithm to understand the real benefits in parallelizing the matching process using MPI-CUDA approach.

A. Experimental setup

For validation of MPI-CUDA algorithm, MPI-CUDA framework has been setup. For this approach, Beowulf cluster of the two nodes is formed. A configuration of an individual node is described below. As each node is equipped with 2 GPUs, event processing is carried out by all 4 GPUs independently. Beowulf is a multi-node cluster used generally for parallel computations. This cluster usually consists of one server node and one or more client nodes connected via Ethernet or some other network. Beowulf cluster is formed as directed in [23].

Table 1: Default Scenario

Parameters in the Default Scenario

Number of events	1000
Attr. per event, min-max	3-5
Number of interf.	10
Constr. per filt., min-max	3-5
Filt. per interf., min-max	22500-27500
Number of names	100
Distribution of names	Uniform
Numerical/string constr.	100% / 0%
Operators	=(25%),≠(25%),>(25%),<(25%)
Number of values	100

A Configuration of the node used to form a cluster is as follows. Core i7-2600 PC, with four cores running at 3.2 GHz, and 16 GB of DDR3 Ram. The Two GPGPUs were a NVIDIA Quadro K600 with 1 GB of DDR3 Ram. We used the GCC compiler, version 4.7, and the CUDA runtime 7.5 for 64 bit Linux.

B. *Latency of Matching*

To evaluate the latency of pure matching, default scenario mentioned in [1] is used whose parameters are listed in Table 1, and used it as a starting point to build a number of different experiments, by changing the various parameters one by one and measuring how this impacts the performance of CCM and MPI-CUDA algorithm.

Default scenario: Table 2 shows the processing times measured by the algorithms under analysis in the default scenario.

Table 2: Processing Time in Default Scenario

Processing Time in the Default Scenario**:**

CCM	MPI+CUDA
0.25245 ms	0.1535 ms

Under this load, if we consider all algorithms, CCM requires 0.25245ms and MPI-CUDA require 0.1535ms, which is a substantial improvement over CCM.

Number of Attributes: Figure 3 shows how performance changes with a number of attributes per event. Higher matching time is required by both the algorithms with an increase in a number of attributes. CCM processes attributes of events, in parallel using GPU. As MPI-CUDA algorithm is based on CCM, it also exhibits similar performance. MPI-CUDA exhibits 1.65X speedup compared to single GPU. But as a number of attributes go on increasing MPI-CUDA exhibits 1.82X speedup. This indicates that MPI-CUDA approach works well for a large number of attributes in events. Eventually, this speedup achieved by MPI-CUDA is low due to more communication overhead.

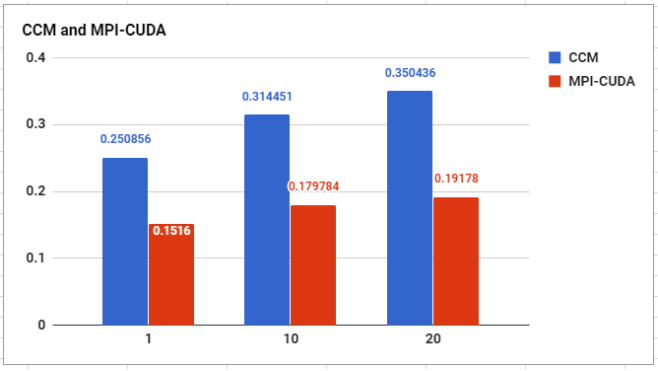

Figure 3. Number of Attributes Per Event.

Number of Events: Figure 4 presents the performance of algorithms in the context of matching time with an increase in the number of events. As this approach focuses on the parallel event processing using a cluster, its performance is validated for a large number of events. Default scenario considered for experiments mentioned in [1] processes 1000 events. As the system under consideration is high-performance event processing system, we expect the events between 1 Lakh to 5 Lakh. The Figure 4 indicates that MPI-CUDA approach requires less time as compared to CCM. Here it is observed that MPI-CUDA approach is suitable for event processing systems where the rate of arrival of an event is a high and large number of events is to be processed by the Pub-Sub system.

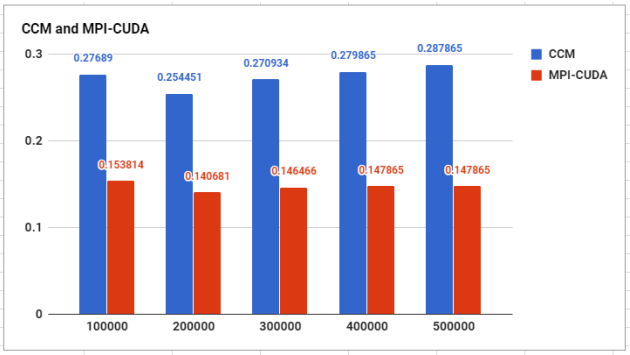

Figure 4. Number of Events for Processing

Number of Constraints: Figure 5 represents the performance of algorithms with a varying number of constraints per filter. The complexity of matching increases with increase in the number of constraints. CCM require more time with an increase in the number of constraints. MPI-CUDA approach works well because parallel processing of events and hence total matching time is improved.

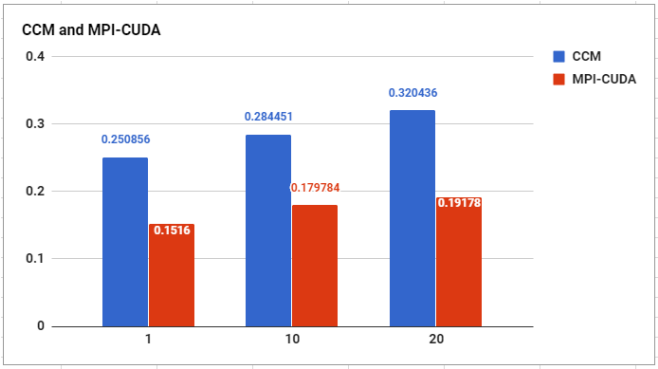

Figure 5 . Number of Constraints per filter

C. *Final Consideration*

We draw some general conclusions from the vast majority of experimentation and observation. Parallel, scalable and high-performance Pub-Sub system can be designed by extending single GPU based CCM algorithm. Parallel event processing is a complete solution for building event-driven applications,

where a rate of arrival of the event is high. Inherent parallelism is involved in the matching process, so it is possible to extend it to multilevel and hybrid parallelism.

We present some remark on the use of multiple GPUs easily available in modern computer architecture and its effective use by CUDA programming. It is possible to achieve high throughput for data intensive application using MPI-CUDA cluster. Due to more processing power, the faster runtime is achieved. MPI-CUDA approach scales across the cluster of computing nodes and so well suited for a processing of large number of events, but communication overhead is the bottleneck in the system. With the increase in the number of attributes from 1 to 10 an average speedup of 1.77X over CCM is observed. Speedup of 2X w.r.t CCM in average matching latency is observed for 500K events. With the increase in the number of constraints from 1 to 20 an average speedup of 1.77X w.r.t CCM in processing time is observed.

ACKNOWLEDGMENT

We sincerely thank Alessandro Margara, and Gianpaolo Cugola for providing code (CCM) for modification as well as allow us to compare results of MPI-CUDA approach. Also, we thank authors who provided data set for experimentation. This work is supported by Walchand College of Engineering, Sangli

REFERENCES

[1] Alessandro Margara, Gianpaolo Cugola *"High-Performance Publish-Subscribe Matching Using Parallel Hardware"*, IEEE Transactions on Parallel and Distributed Systems Volume.25, Issue.1,January 2014

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, *"Design and evaluation of a wide-area event notification service"*, ACM TCS, 2001

[3] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, *"Matching events in a content-based subscription system"*, In PODC, 1999 and evaluation of a wide-area event notification service. ACM TCS, 2001.

[4] H.-A. Jacobsen, A. Cheung, G. Lia, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh, *"The PADRES publish/subscribe system"*, Handbook of Research on Adv. Dist. Event-Based Sys. Pub. /Sub. and Message Filtering Tech., 2009.

[5] Peter R. Pietzuch, *"Hermes: A scalable event-based middleware"*, Technical Report University of GANBRIDGG Computer Laboratory.

[6] T. Yan and H. Garcia-Molina, *"Index structures for selective dissemination of information under the Boolean model"*, ACM TODS,1994.

[7] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, *"Filtering algorithms and implementation for fast pub/sub systems"*, SIGMOD, 2001.

[8] S. Whang, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, E. Veer, R. Yerneni, and H. Garcia-Molina, *"Indexing Boolean expressions"*, In VLDB'09.

[9] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, *"Matching events in a content-based subscription system"*, In PODC, 1999.

[10] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, *"Efficient filtering in publish-subscribe systems using binary decision diagrams"*, In ICSE, 2001.

[11] G. Li, S. Hou, and H.-A. Jacobsen, *"A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams"*, In ICDCS,2003.

[12] G. Cugola and G. Picco, *"REDS: A Reconfigurable Dispatching System"*, In SEM, pages 9-16, Portland, 2006. ACM Press.

[13] Tania Banerjee Mishra, Sartaj Sahni, *"PUBSUB: An Efficient Publish/Subscribe System"*, IEEE Transactions on Computers Volume. 64, Issue.4, 2014.

[14] Mohammad Sadoghi, Hans-Arno Jacobsen, "Analysis and Optimization for Boolean Expression Indexing ACM Transactions on Database Systems", Vol. 38, No. 2, Article 8, 2013.

[15] M. Sadoghi and H.-A. Jacobsen, "BE-Tree An Index Structure to Efficiently Match Boolean Expressions over High-dimensional Discrete Space", SIGMOD, 2011.

[16] A. Farroukh, E. Ferzli, N. Tajuddin, and H.-A. Jacobsen, *"Parallel Event Processing for Content-Based Publish/Subscribe Systems"*, (DEBS '09), pp.8:1-8:4, 2009.

[17] K.H. Tsoi, I. Papagiannis, M. Migliavacca, W. Luk, and P Pietzuch, *"Accelerating Publish/Subscribe Matching on Reconfigurable Supercomputing Platforms"*, Proc. Many-Core and Reconfigurable Supercomputing Conf., 2010.

[18] Medha A. Shah, D.B.Kulkarni, *"Storm Pub-Sub: High Performance, Scalable Content Based Event Matching System Using Storm"*, In IPDPS,W' 2015.

[19] Raphael Barazzutti, Pascal Felber, *"Streamhub: A Massively Parallel Architecture for High-Performance Content-Based Publish/Subscribe"*, DEBS, 2013, Arlington, Texas, USA.

[20] Medha Shah, D.B.Kulkarni, *"Enabling Qos Support for Multi-Core Message Broker in Publish/Subscribe System"*, Advance Computing Conference (IACC), 2014 IEEE International conference.

[21] Zhaoran Wang, Xiaotao Chang, *"Pub/Sub on Stream: A Multi-Core Based Message Broker with Qos Support"*, DEBS, 2012, Berlin, Germany July 2012.Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3 (Mar. 2003), 1289-1305.

[22] A. Carzaniga and A.L. Wolf, "Forwarding in a Content-Based Network," Proc. SIGCOMM, pp. 163-174, 2003.

[23] https://www.linux.com/blog/building-beowulf-cluster-just-13-steps

