Distributed High Performance Computing using JAVA

Subarna Shakya
Department of Electronics and Computer Engineering
Pulchowk Campus,Institute of Engineering
Tribhuvan University,Lalitpur,Nepal
Email: drss@ioe.edu.np

*Ram Sharan Chaulagain / †Santosh Pandey / Prakash Gyawali‡
Department of Electronics ad Computer Engineering
Advanced College of Engineering and Management
Tribhuvan University,Lalitpur,Nepal
Email: *tangent.rams@gmail.com /†santosh.pandey2222@gmail.com / hsageyolk@gmail.com‡

***Abstract*—This research paper is a study on trade-off of using Java to create cluster capable of achieving high-performance computing. Here we have made a distributed computing architecture in Java using Low-level API implementations. Time is crucial in any field of work. We value fast and optic results. Creating cluster base through traditional method using C, C++ or FORTAN programming language can be tedious, complex and time consuming. Distributed High Performance Computing using Java is a vision to achieve high computation performance using Low-level APIs and commodity software development strategy using Java. This paper exhibits the pros and cons of using Java for development of distributed high performance application . The developed architecture supports inter independent tasks parallelism and domain decomposition strategy has been used for partitioning task. The System is based on self-designed distributed architecture without use of any higher abstraction libraries but using low-level abstraction (Java sockets) . The architecture performs the tasks of job division, handling task queue/result queue, scheduling the tasks and supervising the worker nodes. The salient features of our architecture are plug and play software framework, dynamic worker addition and fault tolerance in case of Node failure.**

***Index Terms*—Distributed computing, HPC, Java**

I. Introduction

With the advancement in the field of technology, processors are getting better and better in aspect of performance and speed. But it is uncertain that this trend will continue. The numbers of transistors packing into the single chip is increasing but clockspeed of processers already hit the wall so, computation throughput is increased by using those transistors to pack multiple processors onto the same chip [9].There is high demand of performance in the field of computing which cannot be fulfilled by single processor alone. Here comes the concept of parallel computing including multi-processor system, multi-core system and multi-computer systems.
Many business industries, scientists, researchers prefer multi-computer system over multi-processer system due to its features like low cost, flexibility, scalability etc. This paper is focused on creating an architecture that allows high performance computing to be possible using multi-computer system using Java. In other words, this is a platform for applications that make use of parallel processing and supercomputing.

From its birth till now, Java has emerged as a leading programming language, especially in web-based and distributive computing environment. Considering High Performance Computing, it has established itself as a highly approachable option. The special features of Java [15] like: platform independence, portability, security, object oriented, built-in networking and multi-threading support etc. make it appealing for working in parallel computing. Java being multi-threaded,has several models for distributed and parallel computing with different levels of abstractions allowing us to make choice as we need. The former problem of performance gap between Java and other native language is almost insignificant thanks to Just-in-Time (JIT) compiler of Java Virtual Machine (JVM) that obtains native performance from Java byte-code. It is a portable programming language that can be used on any machine, either homogeneous or heterogeneous machines that support Java. Its rich libraries supports various implementations of distributed computing protocols and communication libraries of which some are mentioned in related work section. So, Java is a good option for developing multi-computer systems or distributed systems. We have to consider task distribution, flexibility of task for parallelizing, communication between computers, congestion in network and other multiple things that affect the performance of any system. These are some of the analyzing criteria for any language to be used in high performance computing [13].

Following above details, an architecture has been designed and implemented using Java. The system is based on Beowulfs cluster [1] where the main server is connected to the worker nodes through Ethernet switch. The architecture is based on divide and conquer strategy for performing computations. The design is made such that the task is first partitioned and mapped between the worker nodes by the server or Job Executor. Furthermore, along with the high computing aspect, it also comes with the package that includes fault tolerance and dynamic worker addition and removal. In other words, if a computer is unable to perform a task given, it is detected

ISBN: 978-1-5090-6471-7/17/$31.00 2017 IEEE

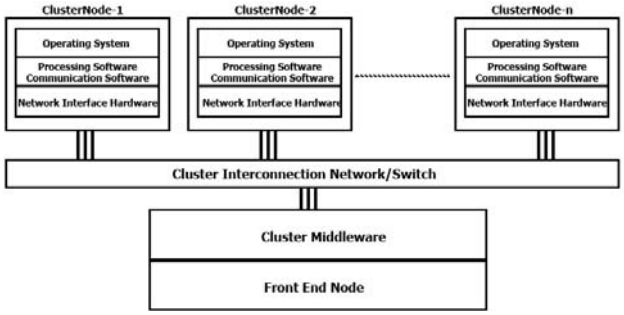


Fig. 1: Architecture of Cluster

and the task is assigned to another idle worker node. Also, the support of dynamic worker means we can add any number of nodes to the system as per our need or task demand during the execution phase of the task. These features are easily achievable using java . Features of java makes it a good choice for heterogeneous computing. Though java has so much benefits for using it to develop distributed computing, it has to be tested for other certain important features like performance, parallel programming model, scaling,scientific computations, data parallelism etc. These features have been tested and analyzed with our architecture described in result and analysis section.

II. Related Works

There are many paths to create cluster. However, the difference lies on the simplicity provided for the developer. Similar works have been done on the field of super-computing with different objectives in mind. We have studied some similar works for information purpose and they have been contributive to our work.

A. Oscar with MPI

OSCAR allows users, regardless of their experience level with a nix environment, to install a Beowulf type high performance computing cluster [4]. It can be used for different implementation purposes as it is a platform for HPC cluster installation. cluster installation.

B. OpenMosix

It was a free cluster management system that provided single-system image (SSI) capabilities. Though its development has been halted by the developers, the LinuxPMI project is continuing development of the former OpenMosix code [11].

C. Others

Although we selected Java sockets for development of both high level libraries and Java parallel applications,various other options of Java has been used for creating HPC at various levels of abstraction . They use collection of libraries for comunication , scheduling , handling data or resource management purposes. Some of the communication libraries are mpiJava [12] , Java Remote Method Invocation(RMI) , Parallel Java etc supporting Shared memory model , MPI-like Message-passing paradigm or both. Parallel Programming language like X10 has also been developed using Java making distributed applications faster and easier programming with libraries. It mainly focuses on supporting parallel programming for more scalability. Various frameworks for distributed computing like hadoop are also developed specialized for big data being able to fulfill the present need of analyzing internet-scale data being scalable and handling large data storage. Several effort are going on for data parallelism using hardware acceleration using GPU(s). Currently hardware accleration can be used using java bindings of various libraries like CUDA(jCUDA [7], java-gpu [6]) and OpenCL (jocl [8]).

III. System Design and Architecture

The system is based on Beowulfs cluster where the main server is connected to the worker nodes through Ethernet switch. The design is made such that the task is partitioned and mapped between the worker nodes by the server and once the worker nodes complete the tasks, the result is available to the server node which integrates the result to provide the user with the final output. The communication delay during the scheduling of task and retrieving of result can be minimized by either using high quality Ethernet switch or using of routing algorithms or by using some different means of communication like InfiniBand, Myrinet, QsNet II etc.[2]

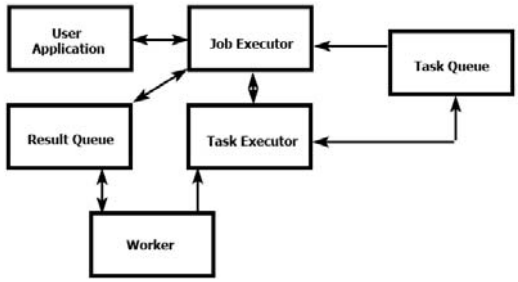


Fig. 2: Software Architecture.

When a worker tries to connect to the Work Server, it is connected through the socket and it is given a specific ID. Then it runs in a different thread in the main Server. The Executor passes the task to the distributed nodes located in the same network. The workers receive the work through the Task Executor and starts the computation of the task. Each node runs multiple worker session parallely for a better resource utilization. When worker finishes the task, it passes the result to the Executor and the Executor writes the result to a Result Queue. After all threads are closed i.e. All worker finishes the work, the Job Executor reads the various values from the result queue and provides the final result to the User. During the working of the Worker, if they crash or fault arises, the Executor detects it and throws exception to the Executor and again writes the task to the Task Queue overcoming the Fault.

The overall working of the cluster is shown in the figure. The user assigns the input to the Job Executor The program forwards small chunks of tasks as per instruction to different nodes such that there is no problem in result or calculation process. The nodes perform task assigned to them. The results are returned and integrated to provide the user with the final result or solution.

A. Nodes

There are mainly two nodes in the architecture . One is the Job Executor node and another is Worker node. As the architecture is based on Java, it runs on both windows and Linux OS. It handles and synchronizes all the tasks in the system. The server acts as a global broker for data flow management to the cluster. For better understanding, the cluster network can be taken as the hardware part of a computer and the server can be viewed as the operating system. So, the operating system is a resource manager and task scheduler. It manages and distributes the work load among the CPUs/nodes in cluster. Another node is an individual computer in the cluster which is a worker. These are normal computers which we use for basic requirements. The reason to use normal computer is to achieve high performance computing in cheap price rather than using expensive mainframe supercomputers. The worker node will be running on a customized version of Linux which will be capable to run only special functions required by the cluster. Every node can be installed with dedicated graphics card for vector processing for hardware acceleration on data parallelism.

B. Memory Architecture

The cluster network uses distributed memory like architecture. Memory locations of task listed are shared to every worker node as in Distributed Global Address Space(DGAS) [3]. All nodes have their own non-shared local memory for processing data. Initially while retrieving data from a network, the server collects all the data and distribute among the nodes. Message passage within process is done using our architecture for distributed computing.

C. Computation Architecture

This cluster is designed for high performance computing which requires parallel execution for faster computation. The designed architecture is specially focused of Single Instruction Multiple Data(SIMD) programming model. Single problem is partitioned into smaller independent tasks to be executed in parallel. For hardware acceleration, GPU(s) can also be added to the cluster for vector processing of the data for increasing the computation of the cluster implementing massive data parallelism. High performance arbitrary precision library was used for achieving higher precision in calculations.

D. Type of Parallelism

Various type of parallelism exists in parallel computing such as bit-wise parallelism, instruction wise parallelism and task wise parallelism. Our architecture uses parallelism in higher abstraction through task parallelism. It allows us to use higher level of parallelism. A single programs data is broken into various independent block of data which has been distributed among the nodes for parallel execution on their local memory. For managing parallel programming, Java threads are used using Java.lang.Thread class and its various methods. Methods like wait() , notifyall() etc are used for communicating between the threads. The join() method of java thread is used to know when all threads from thread pool complete their work for computing the final result. .

E. Data storage and management

Managing data is one of the biggest challenge for distributed computing. Higher availability of data for data intensive program directly affects the performance of the system. The cluster has a distributed memory model. A distributed data storage has higher performance than centralized system like mySQL. Many new distributed data storge system have emerged for completing the requirement of higher performance like Google File System(GFS) [10], Hadoop Distributed File System(HDFS)[5] etc which are based on NoSQL. Currently this architecture doesnt use any standard Database technology. All the required data are stored in the servers memory. For any application, which requires sets of data, the server distributes the data to the remote worker as instructed by the program. The data repository can also be used for inter-node communication between the remote worker nodes.

IV. Parallel Programming Model

The model that we adopted is the Task/Channel Model. It represents a parallel computation as a set of tasks that may interact with each other by sending message through channel. A task is a program, its local memory, and collection of I/O ports. A task sends local data values to other tasks via global communication and conversely receive inputs too. A channel is a message queue that connects one tasks output port with other tasks input port via server mapper. Under Task/Channel model, our design methodology is slightly different than Fosters Design Methodology [14]. Fosters Design consists of four stages, partitioning communicating, agglomeration and mapping whereas we adopted partitioning, queuing and mapping stages.

1) Partitioning: Partitioning includes division of computation and data into pieces. The problem is decomposed into number of primitive tasks. A good partitioning splits both the computation and data into many small pieces. Generally, partition is done through data centric or computation centric approach.
2) Queuing: Queuing is the stage where partitioned blocks are queued into the array list.Every blocks are indexed with array index which is further used as key for mapping.Its is accesed by workers(clients) with mapper.
3) Mapping: Mapping is the process of assigning tasks to processors. In a cluster based model, manual or program aided mapping is needed. The main goal of mapping

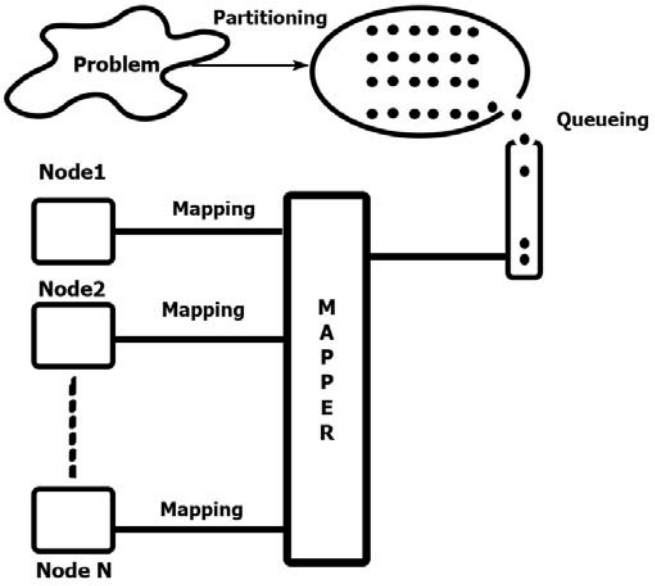


Fig. 3: Parallelism

is to maximize processor utilization and minimize inter processor communication. Mapper assigns the blocks to processors in cluster network.Workers can randomly access the task queue from mapper .The channel we If we are executing our program on a centralized multiprocessor, the operating system automatically maps processes to processors.

Domain decomposition is the parallel algorithm approach in which, we first divide data into pieces and then determine how to associate computations with data. Typically, our focus is on the largest and/or most frequently accessed data structure.

Strategy involved: Block allocation

Suppose there exist n numbers of elements and p numbers of processors then we can allocate n/p numbers of blocks.

First element controlled by node i= i$*$ $(\frac{n}{p})$

Last element controlled by process i= [(i+1)$(\frac{n}{p})$]-1

Processes controlling particular array element j is $[\frac{p(j+1)-1}{n}]$

V. Test Case

Various test were performed on the designed architecture for evaluating its performance using numerical computation. Test case were designed to evaluate the constraints which determines the efficiency and performance of the architecture. We used CPUs of i3 processor at 3.30 GHz speed and 2GB RAM. The hardware specification of our server is intel i5 processor @ 2.20 Ghz (4CPUs) and 4GB RAM have L1 cache of 512KB and L3 cache of 3MB. The connection was maintained by 100MBps Ethernet. The operating system was varied among Windows (8.1) and Linux (Ubuntu 16.6) and JDK (1.8.0.81), JVM (Server Edition).

For testing our architecture, we ran a parallel program for calculating numerical integration using Simpsons 3/8 rule which calculates area under curve. The motive for selecting integration as test case is because of its simplicity for testing with varying number of steps. Apfloat variable were used for testing using 26-bit precision which provided higher arbitrary precision as required. We benchmarked parallel program on our architecture by executing it multiple times on multiple numbers of nodes and cores selection. For each number of processors, we compute mean execution time for the completion of the numerical computation. The experimental result is then compared in terms of performance.

VI. Result and Analysis

The tests were conducted on 1-16 nodes at computer lab. After completing the setup, we tested the cluster in different scenarios where numbers of nodes were connected at a time with specific number of cores utilized. Every computer was connected to a network switch along with a router in star topology. CAT5 cables were used as Ethernet connection for the network. IP address of each computer was assigned by the switch. Various experiments were performed using various number of nodes , threads and task size. Various constraints affects the performance of the system like network latency , no. of cores used , size of chunk depending on resource availability etc. Maximum numbers of threads on a single node was limited to 4 as the desktop would only have 4 cores while enabling Hyper-Threading mode. Same version of JVM were used for the experiment. Following results were obtained while experimenting with different nodes and different threads on single node.

A. Windows vs Linux

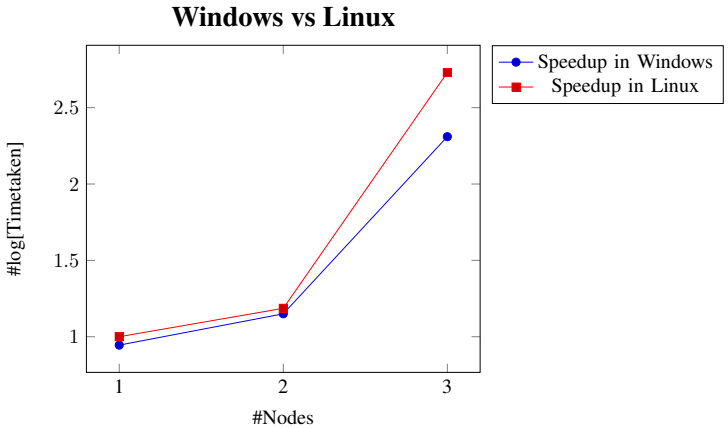


Fig. 4: Windows Vs Linux

Comparison of performance for the designed architecture between windows and Linux Operating System (OS) was made on same node with 4 threads on both machine. For Windows,

Windows 10 was used and for Linux, Ubuntu 14.04 was used. From the following Figure 4 plotted between no. of nodes and speedup achieved, we can see that Linux has more speedup than windows in the basis of execution time. The test included same chunk size and same machines. Linux is always preferred over windows for higher performance and we can see the result. We can customize the Linux kernel as required for our designed architecture to achieve higher performance. But doing same with windows is difficult. So using linux for our distributed architecture was best choice for us.

B. Same no. of nodes with varying chunk size

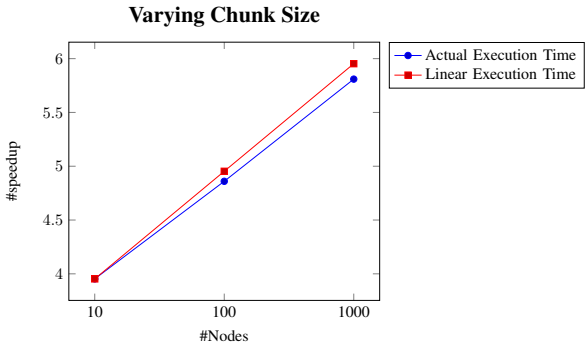


Fig. 5: Varying Chunk Size

This time, chunk size was differed on the same machines for studying the effect of chunk size on performance. For experiment, 7 nodes with 4 threads on each machines were used. The graph of Figure 5 shows log value of execution time on basis of size of chunk for comparision. Red lines shows the linear increase in time with respect to chunk size whereas blue line shows actual execution. We can interpret that the execution time doesnt increases linearly though chunk size increases linearly. Performance of the system increases while working with large chunk size. One of the reason for this increased performance is due to reduce in network time as chunk size are large and no. of chunks are less which is briefly defined in Chunk Size determination section.

C. Core vs Node

This experiment was performed for analyzing impact on performance of system due to network latency of the architecture based of Java sockets for communication. Accessing data from network takes longer time than accessing it from Primary memory. On first run , numerical computation was performed on single node with 3 threads. On next run, 3 nodes with single thread on each were used. Total no. of threads or worker and chunk size were same on both run. Figure 6 shows the result of core vs node comparison. We can see effect of network latency on the performance of the system. For efficient performance, network time should be reduced as much as possible. It can be done by optimizing the size of chunk and no. of partition of chunks.

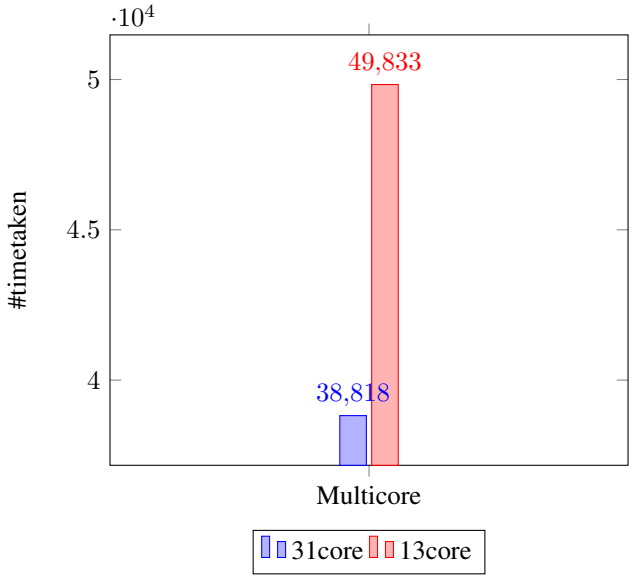
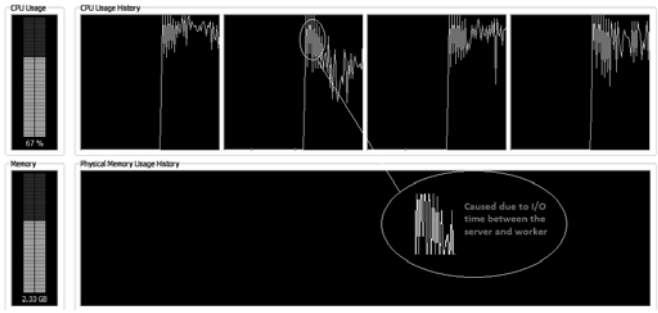
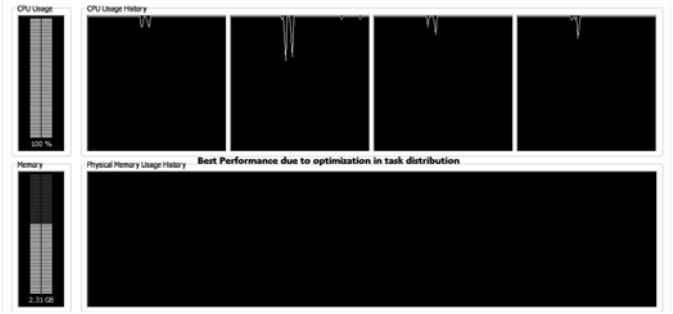


Fig. 6: Core vs Node



(a) Gitter



(b) best performance

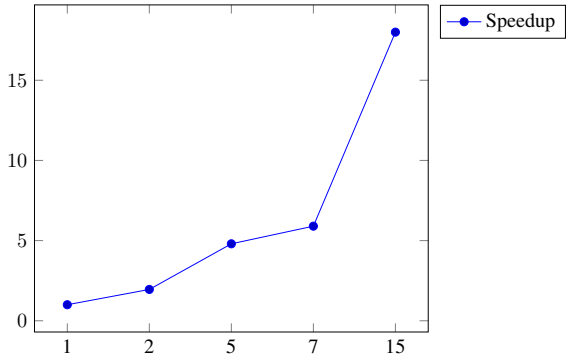
Fig. 7: Performance

D. Chunk Size Determination

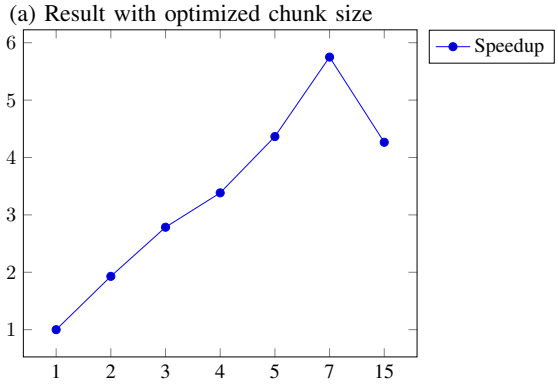
From above experiments, we can conclude that determining the size of chunk or an independent block of data is crucial as it affects network latency and CPU idle time. Chunk size should be optimized according to the resources available on the cluster. In our architecture ,we have manually selected optimum value for the chunk size as we have a dedicated cluster network. Performing automatic chunk size determination is also possible using various cluster development tools

like MPI or PVM. Resource management tools for cluster are used for scheduling task and allocating resources efficiently to nodes. If selected chunk size is much small , then no. of chunk will be large and more network I/O is performed as shown in Figure 7a. The fig shows how I/O time affects performance for computation as CPU cycles are used more for I/O than calculation. It degrades the efficiency of the system. Whereas if selected chunk size is too large, then no. of chunk will be less and some processor may remain idle . So optimum value of chunk size is important.Figure 7 shows the CPU cycle of 4 cores of a worker node when chunk size is optimized.

E. Optimized output from Designed Architecture



(a) Result with optimized chunk size



(b) Result with small chunk size

Fig. 8: Optimized output from Designed Architecture

After analyzing above results, some tweaks were made on the architecture for making it more efficient. Figure 8a shows the performance of the architecture using 15 nodes. Chunk size were selected in such a way that no worker threads remained idle for much time. The actual speedup of the system is compared with linear increase. Exponential speedup is obtained from the architecture as stated by Amdahl's law for inter-independent tasks. Figure 8b shows how the performance of cluster decreases if chunk size is not allocated properly.

VII. Conclusion

From the above observations, we can conclude that using low-level API for creating HPC may be complex in case of job scheduling and automatic resource management. When it comes on distributed HPC, Java could outrun other languages in case of higher level of abstraction. It can be a good choice for developing distributed computing due to its features like portability, GUI (Graphics User Interface) development, in-built multi-threading and network library. Heterogeneity can be easily dealt with Java. Using Java or not for HPC is choice of the user. If the requirement includes portability, high availability, easier software development, GUI or security Java will be the best option for that type of applications. With integration of Nosql based Database technology, this architecture can be used to process huge datasets and will help the system to be scalable. This architecture can be made more modular for integrating and developing other applications making it more useful.

Acknowledgment

We would like to thank our supervisor Prof. Dr. Subarna Shakya for his diligent effort to help and guide us along our journey and providing us with helpful insights to reach this goal. We would also like to remember our friends and family as they too have supported us to reach this goal.

References

[1] Beowulf Clustering frequently asked questions. www.beowulf.org/overview.faq.html. Accessed: 2016-09-03.
[2] Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. http://www.cloudbus.org/papers/ic_cluster.pdf. Accessed: 2016-08-13.
[3] Distributed Shared Memory distributed global address space(dgas). https://en.wikipedia.org/wiki/Distributed_shared_memory. Accessed: 2016-09-03.
[4] Edgewell software. (n.d.). oscar project. http://svn.oscar.openclusergroup.org/trac/oscar. Accessed: 2016-08-11.
[5] The hadoop distributed file system (hdfs). https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Accessed: 2016-09-03.
[6] javagpu. http://code.google.com/p/java-gpu. Accessed: 2016-08-13.
[7] jcuda. http://hoopoe-cloud.com/Solutions/jCUDA/Default.aspx. Accessed: 2016-08-13.
[8] jocl. http://jocl.org. Accessed: 2016-08-13.
[9] Moore's law and multicore. http://web.engr.oregonstate.edu/~mjb/cs475/Handouts/moores.law.and.multicore.2pp.pdf. Accessed: 2016-07-25.
[10] Sanjay ghemawat, howard gobioff, and shun-tak leungThe Google File System. https://research.google.com/archive/gfs.html. Accessed: 2016-09-03.
[11] Sourcr forge. (n.d.). openmosix, an open source linux cluster project. http://openmosix.sorceforge.net. Accessed: 2016-08-13.
[12] J. Spacco B. Pugh. *mpiJava, MPJava :High-Performance Message Passing in Java using Java.nio*. Proc. 16th Intl. Workshop on Languages andCompilers for Parallel Computing (LCPC03), LNCS vol. 2958, College Station, TX, USA, 2003, pp. 323339.
[13] Kevin Dowd Charles Severance. *High Performance Computing, 2nd Edition*. O'Reilly Media, July 1998.
[14] Michael J. Quienn. *Parallel Programming in C with MPI and OpenMP,1st Edition*. McGraw Hill Education, 2003.
[15] Herbert Schildt. *JAVA: The Complete Reference, 8th Edition*. McGraw Hill Education, Aug 2011.