© 2015 IJSRSET | Volume 1 | Issue 4 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Recent Trends comprehensive survey of Asynchronous Network and its Significant

Ajitesh S. Baghel, Rakesh Kumar Katare
Department of Computer Science, A. P. S. Univesity, Rewa Madhya Pradesh, India

ABSTRACT

Advancement in electronics and computer architecture has opened new domains of the parallel and distributed computing. The advent of the Multi Core CPU's with the blending of the open MPI techniques has given the wings to the distributed computing with assurance of the parallelism. In this proposal, various important aspects of asynchronous algorithms and its data structures for parallel and distributed architecture will be investigated. This article has proposed and will examine networks of processor for asynchronous system to compute faster for more iteration. The complexity of interprocessor communication will be investigated. Hence efficient asynchronous algorithm is main concerned of the study for MPI systems.

Keywords: Asynchronous Network, Distributed System, Parallel Computing

I. INTRODUCTION

Advancement in electronics and computer architecture has opened new domains of the parallel and distributed computing. The advent of the Multi Core CPU's with the blending of the open MPI techniques has given the wings to the distributed computing with assurance of the parallelism. Evolution and 3Gand 4G with distributed functionality with help of the mobile agent's technology has proof the significance of the autonomy of the software code i.e. distributed nature of the agents.

Over the past few years, there have been tremendous efforts to improve the speed and sophistication of large-scale data-parallel processing systems [4]. Practitioners and researchers have built a wide array of programming frameworks [10, 11, 12, 13, 14, 15] and storage systems [16, 17, 18, 19, 20] tailored to a variety of workloads. As the performance of many of these systems is I/O bound, traditional means of improving their speed is to cache data into memory [8, 11]. While caching can dramatically improve read performance, unfortunately, it does not help much with write performance. This is because these highly parallel systems need to provide fault tolerance, and the way they achieve it is by replicating the data written across nodes. Even replicating the data in memory can lead to a significant drop in the write performance, as both the latency and throughput of the network are typically much worse than that of local memory.

Slow writes can significantly hurt the performance of job pipelines, where one job consumes the output of another. These pipelines are regularly produced by workflow managers such as Oozie [18] and Luigi [19], e.g., to perform data extraction with Map Reduce, then execute a SQL query and then run a machine learning algorithm on the query's result. Furthermore, many high-level programming interfaces [20, 21, 22], such as Pig [23] and Flume Java [24], compile programs into multiple Map Reduce jobs that run sequentially. In all these cases, data is replicated across the network in-between each of the steps.

The recent years encountered the development of software standard for clusters computing. The complexity of interprocessor communication in case of asynchronous networks efficiency becomes a course of studies for computer scientists and Mathematicians. The computer scientists are try to investigate new ways to optimize number of links and efficient message passing mechanisms.

The major constraints in asynchronous networks of processors are:

- Load Balancing.
- Fault Tolerance/ Robustness.
- Low Latency
- High Bandwidth
- Energy Efficiency.

The emergence of distributed computation demands the development of new architecture to counter the challenge of new technology.

According to [5], in sequential computing, computability is understood through the Church-Turing's thesis (namely, anything that can be computed, can be computed by a Turing machine). Moreover, when considering the notion of a universal algorithm encountered in sequential computing, such an algorithm "has the ability to act like any algorithm whatsoever. It accepts as inputs the description of any algorithm A and any legal input X, and simply runs, or simulates, A on X. [...] In a sense, a computer [...] is very much like a universal algorithm [25].".

Hence, the question: Is it possible to design a universal algorithm/machine on top of an asynchronous crash-prone distributed system? As we are about to see, it happens that, due the environment (asynchrony and process failures) of a distributed system, and the fact that it cannot control it, distributed computability has a different flavor than computability in sequential computing. Moreover, this is independent of the fact that the communication is by read/write registers or message-passing. Due to its very nature, distributed computing requires cooperation among the processes. Intuitively, the computability issues come from the fact that, due to the net effect of asynchrony and failures, a process can be unable to know if another process has crashed or is only slow (or equivalently if the channel connecting these processes is slow). Moreover, this is true whatever the individual power of each process. To cite "It follows that the limits of computability reflect the difficulty of making decisions in the face of ambiguity, and have little to do with the inherent computational power of individual participants".

Rest of the article is organized as follow, Section II presents the brief introduction about the distributed and parallel computing and its evolution, Section III discusses the recent contribution in the asynchronous computing networks and its significant application domain. Section IV presents proposed research idea and the motivational factor. And finally Section V concludes the papers with the future directions of this work.

II. METHODS AND MATERIAL

Parallel & distributed computing:

Term Network is the simplest word having the highest complexity hidden inside. Modern era is the age of computation; everything is depends, operated and derived from computation. Evolution of the fast computer's and CPU makes life easier, even though this nano-integrated deice designing is one of the complex and adept task. Abstraction is the methodology of the software engineering to hide irrelevant details (complexity) from the end users. Abstraction with felling of uni-user is also termed as transparency in distributed system.

According to Coulouris defines a distributed system as "A system in which hardware or software components located at networked computers communicates and coordinates their actions only by message passing". Whereas according to Tanenbaum defines it as "A collection of independent computers that appear to the users of the system as a single computer".

Leslie Lamport, a famous researcher on timing, message ordering and clock synchronization in distributed systems once said that "A distributed system is one on which I cannot get any work done because some machine I have never heard of has crashed", reflecting on the huge number of challenges faced by distributed system designers.

The nodes in a distributed system are connected by an interconnection network. The communication in between nodes in the distributed system takes place by exchanging messages. Therefore these distributed systems are commonly known as message passing distributed systems with contrast to shared memory communication, which is extensively followed in various multiprocessor and parallel systems. Some of the distributed systems such as wireless ad-hoc networks follow an arbitrary network topology, where the nodes are randomly deployed in the environment. Other kind of distributed systems such as electronic automotive systems are extensively used in real time applications.

Parallel System

It is an "A collection of processing elements that communicate and cooperate to solve large problems fast". Parallel computers or systems are tightly coupled in nature as per Flynn's classification.

Distributed System

It's an "A collection of independent computers that appear to its users as a single coherent system."

A parallel computer is implicitly a distributed system. [Wiki]

A distributed system is composed of a set of machines which do not share a global clock, the machines communicate with each other by exchanging messages over a communication network. Each machine in the distributed system has its own memory and runs its own operating system. The machines in a distributed system offer their resources for collective computation. The resources owned and controlled by a machine are said to be local to it, while the resources owned and controlled by other computers and those that can only be accessed through the network are said to be remote. These resources can be of various types such as computation nodes, storage devices etc. A large number of applications have been developed to harness the power of distributed systems.

Typically a distributed system has the following characteristics:

- **Multiple nodes** – A distributed system is composed of multiple independent nodes belonging to different computers, not merely multiple processors on the same computer.
- **Heterogeneity** – The nodes in a distributed system may consist of machines having different architectures and possibly running different types of operating systems.
- **Message passing** – Processes on the different resource nodes may communicate using diverse networking protocols over different networking technologies. Therefore, the characteristics of the underlying communication links can be different. The nodes in most distributed systems are reachable from one another.
- **Concurrency** – Each of the nodes in a distributed system provides independent functionality, and operates concurrently with other nodes

  Decentralized control – No single computer is necessarily responsible for configuration, management, or policy control for the whole distributed system. However, some functionality may reside in a central node or a set of nodes by necessity.
- **Openness** – Many distributed systems are open, i.e., an unbounded number of nodes or components can be added or changed even while the system is running.

The main objective of a distributed system is to achieve high throughput for distributed applications through concurrent computation and to increase accessibility to resources not commonly available to a single machine.

According to author [5], Distributed computing was born in the late seventies when researchers and engineers started to take into account the intrinsic characteristic of physically distributed systems [26]. Distributed computing arises when one has to solve a problem involving physically distributed entities (called processes, processors, agents, actors, sensors, peers, etc.), such that each entity (a) has only a partial knowledge of the many input parameters of the problem to be solved, and (b) has to compute local outputs which may depend on some non-local input parameters. It follows that the computing entities have necessarily to exchange information and cooperate [27].

Distributed System - A (static) distributed system is made up of n sequential deterministic processes, denotedp1,...,pn. These processes communicate and synchronize through a communication medium, which is either a network that allows the processes to send and receive messages, or a set of atomic read/write registers (atomic registers could be replaced by "weaker" safe or regular registers, but as shown in [28] – where these registers are defined– safe, regular and atomic registers have the same computational power).

Deterministic means here that the behavior of a process is entirely determined from its initial state, the algorithm it executes, and –according to the communication medium– the sequence of values read from atomic registers or the sequence of received messages (hence, obtaining different sequences of values or receiving messages in a different order can produce different behaviors).

Asynchronous Read/Write or Message-Passing System - In an asynchronous (also called time-free) read/write system, the processes are asynchronous in the sense that, for each of them, there is no assumption on its speed (except that it is positive). If the communication is by message-passing, the network also is asynchronous, namely, the transfer duration of any message is finite but arbitrary.

Synchronous Message-Passing System - Differently, the main feature of a synchronous system lies in the existence of an upper bound on message transfer delays. Moreover, (a) this bound is known by the processes, and (b) it is assumed that processing durations are negligible with respect to message transfer delays; consequently processing are assumed to have zero duration.

This type of synchrony is abstracted by the notion of round-based computation. The processes proceed in rounds during which each process first sends messages, then, receive messages, and executes local computation. The fundamental assumption which characterizes a synchronous message-passing system is that a message sent during a round is received by its destination process during the very same round.

Process Crash Failure - The most common failure studied in distributed computing is the process crash failure. Such a failure occurs when a process halts unexpectedly. Before crashing it executes correctly its algorithm, and after having crashed, it never recovers. Let t be the maximal number of processes that may crash; t is a model parameter and the model is called t-resilient model. The asynchronous distributed computing (read/write or message-passing) model in which all processes, except one, may crash is called wait-free model. Hence, wait-free model is synonym of (n−1)-resilient model.

The Notion of Environment and Non-determinism – The environment of a distributed system is the set of failures and (a) synchrony patterns in which the system may evolve. Hence, a system does not master its environment but suffers it. As processes are deterministic, the only non-determinism a distributed system has to cope with is the non-determinism created by its environment.

Complexity vs. Computability Issues - Computability and complexity are the two lenses that allow us to understand and master computing. The following table presents the main issues encountered in distributed computing, when considering these two lenses.

Failure	**Synchronous** free complexity	**Asynchronous** Complexity
Failure-prone	complexity	computability

Advantages:

- **Higher price/performance ratio –** By interconnecting powerful workstations with high speed communication network we achieve higher performance at lower cost.
- **Resource sharing –** A node in a distributed system can access both software and hardware resources of another node remotely over the communication network.
- **Improved availability –** A distributed computing system provides improved reliability and availability because a few components of the system can fail without affecting the availability of the rest of the system.
- **Improved reliability –** By replicating data and services the distributed systems can be made fault tolerant Distributed systems have been used for a wide variety applications ranging from scientific simulations collaborative engineering, supercomputer enabled scientific instruments, applications in Geographical Information Systems (GIS) like weather prediction, railway or airline reservation systems etc.

Distributed systems are composed of multiple computing resources connected by communication links. Since failure of nodes and links are assumed to be independent, larger the system, higher is its probability of failure. Therefore, in a distributed system, failures are relatively common events. Distributed systems should remain at least partially available and functional even if some of their nodes or communication links fail or misbehave. Without fault tolerance mechanisms, the system and the applications running on it need to be restarted every time a failure occurs. Many of the distributed applications mentioned above are long running, taking hours or even days in some cases to complete. If a fault occurs in the middle of a long

running application, long hours of useful computation will be lost. Thus an application may take a long time to complete in the presence of such failures. Fault tolerance techniques can allow applications to run to completion in the presence of faults with minimal disruption. Since such techniques do not need an application to restart when a fault occurs, they also save system resources and improve system throughput.

Distributed Computing:

A concurrent object is an object that can be accessed by several processes. Let us consider a concurrent object Z defined by a sequential specification on a set of total operations. An operation is total is, when executed alone, it always returns a result. A specification is sequential, if all the correct behaviors of the object can be described by sequences of operations. The notion of universality we are interested in concerns the possibility to implement any concurrent object such as Z, despite asynchrony and crashes. If it exists, such an implementation, which takes the sequential specification of Z as input and builds a corresponding concurrent object, is called a universal construction. This is depicted in Fig. 1.

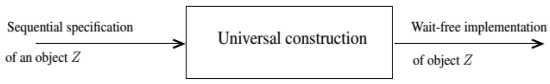

Figure 1. from a Universal Specification to Wait Free Specification depicted by [5]

In some cases the object Z encapsulates a service which can be abstracted as a state machine. A replication-based universal construction of such an object Z is usually called a state machine replication algorithm [39]. Let us remark that the object Z could also be a Turing machine.

The Consensus Object - A consensus object is a one-shot concurrent object defined by a sequential specification that provides the processes with a single operation denoted propose (v) where v is an input parameter (called proposed value"). "One-shot" means that, given a consensus object, a process invokes at most once the operation propose ().

If it terminates, the operation returns a result (called "decided" value). This object can be defined by the three following properties.

- **Validity -** If a process decides a value, this value has been proposed by a process.
- **Agreement -** No two processes decide different values.
- **Termination -** An invocation of propose () by a process that does not crash terminates.

Consensus-Based Universal Construction - Several universal constructions based on atomic registers and consensus objects have been proposed, e.g., [29]. In that sense, and as depicted in Figure 2, consensus is a universal object to design wait-free universal constructions, i.e., wait-free implementations of any concurrent object defined by a sequential specification. This is depicted in Fig. 1.2

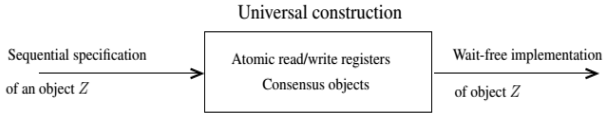

Figure 1.2 Universal construction from atomic registers and consensus objects [5]

Recent trends in Asynchronous Networks:

In a universal construction, consensus objects are used by the processes to build a single total order on the operation invocations applied to the constructed object Z. This is the method used to ensure that the internal representation of Z remains always consistent, and is consequently seen the same way by all processes.

One of the most important of the theoretical results of distributed computing is the celebrated FLP result (named after its authors Fischer, Lynch, and Paterson) [30]), which states that no binary consensus object (a process can only propose v $\in$ {0,1}) can be built in an asynchronous message-passing system whose environment states that (even only) one process may crash.

To prove this impossibility result, the authors have introduced the notion of valence associated with a global state (also called configuration). Considering binary consensus, a global state is0-valent (1-valent) if only 0(1) can be decided from this global state; 0-valent and1-valent states are univalent states. Otherwise, "the dice are not yet cast", and any of0or1can be still decided. This is due to the uncontrolled and unpredictable behavior of the environment (i.e., asynchrony and failure pattern of the considered execution). A decision step of a construction is one that carries the construction from a bivalent state to univalent state. The impossibility proof shows that (a) among all possible initial states, there is a bivalent state, and (2) among all possible executions in

all possible environments, there is at least one execution that makes the construction always progress from a bivalent state to another bivalent state. It is easy to see that, the impossibility to implement a consensus object is related to the impossibility to break non-determinism (i.e., the impossibility to ensure that, in any execution, there is eventually a transition from a bivalent state to a univalent state).

This message-passing result has then been extended to asynchronous systems in which processes communicate only by reading and writing atomic registers instead of sending and receiving messages [29] [31].

Sequential vs. Distributed Computing:

It follows from the previous impossibility results that a network of Turing machines, that progress asynchronously and where at most one may crash (which are two reasonable assumptions) connected by a message-passing facility, or a read/write shared memory, is computationally less powerful than a single reliable Turing machine. As announced in the first section, this shows that the nature of distributed computability issues is different from the nature of Turing's computability issues, namely, it is not related to the computational power of the individual participants.

In this proposal various important aspects of asynchronous algorithmic model of Parallel and Distributed architecture will be suggested. The Term "Asynchronous" means "involving or requiring a form of computer control timing in which a specific operation is begin upon receipt of an indication (signal) that the preceding operation has been completed". We will allow some processors to compute faster and execute more iteration, than other processors, to communicate more frequently than others and we allow the communication delays to substantial and unpredictable. We also allow the communication channels to deliver messages out of order.

Whereas Author [6] has talked about the asynchronous computing related to distributed computing. According to author – The ever growing amount of data produced by parallel numerical simulations calls for new practices to reduce the pressure on I/Os. For instance, the complete chemical structure of the cap Sid of the HIV-1 virus has recently been resolved [32]. The molecular model has a total of 64 million atoms. To simulate this model, scientists used the Blue Water supercomputer, the simulation producing about 10To per run of 100 nanoseconds of simulated time, which makes the analysis of the trajectory very difficult.

Instead of saving raw data to disks for further post processing, the in situ analytics paradigm proposes to perform data processing as closely as possible to where and when the data are produced [33]. The goal is first to reduce the amount of data to be transferred and stored, but also to parallelize analytics on the large supercomputer booked for the simulation. Authors [6] approach also enables to get a live feedback on the current simulation state, and, if necessary, to take early measures to stop the simulation or change some parameters [34].

These processing workflows being interleaved with the simulation, the ease of use, flexibility as well as the overall performance impact must be carefully considered. We can distinguish different mappings for analytics, adopting the vocable of [35]: in situ embedded in the simulation code, or running asynchronously on the same nodes but often on dedicated helper cores; in transit on staging nodes dedicated to analytics; or more classically once data have been saved to disk. Depending on the application domain and the analytics algorithms, the needs can range from simple filtering schemes, for instance removing the water atoms before saving a time step of a molecular dynamics simulation, up to producing high quality images [33].

Next directional approach has been adopted by [7], In this big data era, the data size is growing at an unprecedented scale. From videos in Youtube, security footage at airports to astronomical data collected at the large synoptic survey telescope, tons of data are being generated everyday everywhere. In a recent digital universe study by EMC, the world created about 1.8 zeta bytes of data in 2011. Facebook alone, for example, is estimated to be creating 12 terabytes of data every day. The amount of data across the globe is also expected to double every two years, and will reach 35 zeta bytes by 2020.

According to [7], to alleviate this big data problem, the use of stochastic techniques has recently drawn a lot of interest. Most of them are based on variants of the stochastic gradient descent (Shalev-Shwartz et al., 2007)

[36]. Authors [7] idea is to replace the gradient over the whole data set by the gradient at a single sample (or over a small mini-batch of samples). Hence, its per-iteration complexity is much lower, and can scale to much larger data sets. While the stochastic approach alleviates the big data problem by processing only a small sample subset in each iteration, an alternative is to use distributed processing. This is particularly natural for many big data applications, in which the data sets are too large to be stored or processed on one single computer. In distributed optimization algorithms, communication among the computing nodes is based on either shared memory (Niu et al., 2011[37]) or distributed memory (Langford et al., 2009 [40]; Agarwal & Duchi, 2011 [41]; Ho et al., 2013 [42]; Li et al., 2013 [41]). In this paper [7], authors have focused on algorithms using distributed memory, as they can often handle much larger data sets. Consider minimizing a function f (x) in a distributed computing environment with N nodes. Assume that this function can be decomposed into N components as

$$f(x) = \sum_{i=1}^{N} f_i(x),$$

Where, each fi is a local objective involving only the data subset residing on node i. This type of problems is often encountered in various areas such as machine learning, signal processing and wireless communication (Bertsekas & Tsitsiklis, 1989 [42]; Zhu et al., 2010 [43]). For example, in regularized risk minimization, x is the model parameter to be estimated, and fi is the regularized risk functional defined on the data subset at node i.

The minimization of f (x) can be reformulated as the following global variable consensus optimization problem (Boyd et al., 2011; Bertsekas & Tsitsiklis, 1989 [42]):

$$\min_{x_1,\ldots,x_N,z} \sum_{i=1}^{N} f_i(x_i) \ : \ x_i = z, \ i = 1, 2, \ldots, N,$$

Where, z is the so-called consensus variable, and xi is node i's local copy of the parameter to be learned. In a distributed computing environment, this problem can be efficiently solved by the alternating direction method of multipliers (ADMM) algorithm (Boyd et al., 2011), which has been popularly used in various areas such as machine learning, computer vision and data mining. Essentially, one of the nodes, called the master, is responsible for updating the consensus variable z, while the remaining nodes are called workers. Each worker minimizes its local objective fi (in parallel) based on its data subset; and sends the updated local copy xi to the master. The master, in turn, updates z by driving the x i 's into consensus, and then distributes the updated value back to the workers, and the process re-iterates.

In this proposed work we will try to investigate the communication complexity of various aspects of asynchronous network. Through the properties of Interconnection Network algorithm will be proposed for fine tuning of the performance of Parallel and Distributed Systems. Fault tolerance and Load balancing the two major aspects of parallel computing will also be compared against conventional algorithms.

Through the topological properties of Interconnection Network some algorithms will be proposed for fine tuning the performance of Parallel and Distributed Systems. Fault tolerance and Load balancing the two major aspects of parallel computing will also be compared against conventional algorithms.

Problem formulation & proposed idea:

In the recent years of study in the field of computer science, we have seen an explosion of interest in asynchronous networks of processors for parallel and distributed systems. From theoretical point of view this work has provided a challenging range of problem with new ground rules for the study of various asynchronous algorithmic models.

D. Chazan and W. L., Miranker [45] was introduced "Asynchronous algorithmic models" in 1969, in their paper 'Chaotic relaxation Linear Algebra & Application', in the context of iterative solution of linear systems of equations. This model has also subsequently studied by several others authors. In 1978, M.G. Baudet [46] presents study of 'Asynchronous iterative methods for multiprocessors' and in 1982 D.P. Bertsekas [47] present this paper for 'Distributed dynamic programming'.

The Asynchronous convergence theorem is presented by D.P. Bertsekas [47], in his paper," Distributed asynchronous computation of fixed points" in the year 1983. Necessary condition for asynchronous

convergence is discussed in 1987, by J.N. Tsitsiklis [48] in his paper 'On the stability of asynchronous iterative processes'. Mathematical system theory is new branch of study for asynchronous networks of processors for parallel and distributed systems. The asynchronous relaxation methods for differential equations they proposed in 1987 by D. Mitra in 'Asynchronous relaxations for the numerical solution of differential equations by parallel processors'. B. Lang., J.C. Miallov and P. Spiteri have studied asynchronous algorithms for two-point boundary value problems arising in optimal control in 1986 in 'Asynchronous relaxation algorithms for optimal control problems', math. & compute. simul. F. Robert studied "Totally asynchronous relaxation" in year 1976.

The observation that the invariant distribution of a Markov chain can be found by totally asynchronous algorithm after fixing the value of a single coordinate of the distribution vector is new at that time. Totally asynchronous relaxation methods involving monotone mappings were studied in 1982 by D.P. Bertsekas [47] paper 'Distributed dynamic programming'.

The above given network performance parameters depend not only on the network architecture but also on a number of factors relating to application and their data exchange characteristic. The challenge in interconnection network design is finding the right match between communication need of applications on one side and capabilities and limitations inherent in each architecture on the other hand. This, in turn, explains the proliferation of implemented and proposed connectivity, sometimes characterized as interconnection network see in B. Parhami and M.A. Rakov paper for 'Perfect difference networks and related interconnection structures for parallel and distributed systems'.

In this proposal, various important aspects of asynchronous algorithms and its data structures for parallel and distributed architecture will be investigated. We will examine networks of processor for asynchronous system to compute faster for more iteration. The complexity of interprocessor communication will be investigated. We will development P-RAM algorithms. The algorithms for the following characteristics of asynchronous network of processors will be continued:

- Load balancing.
- Fault tolerance/ Robustness.
- Low latency
- High Bandwidth
- Energy Efficiency

The connectivity of Processor's will be explained in terms of the properties of topology & projective geometry. As we know that the study of Asynchronous processors of network required the study of protocol to stream time of the processors therefore the protocol will be investigated for the development of algorithms. We will use graph theory and set theory as a basic tools to study the asynchronous network model. The algorithm for data routing and network flow model will be theme of studied in future work. In the next step we will also study load balancing algorithm for asynchronous system. The communication complexity of various asynchronous networks of processors will also be investigated. We will also develop algorithms and data structure for linear array of processors for asynchronous systems of processors. Analysis of Parallel iteration and recursion will also be the center of study in this proposal.

III. RESULTS AND DISCUSSION

Proposed idea and Motivation:

The goal of this research is:

We will follow the following methodology to achieve the results:

- First, the proposed system surveyed the existing methods related to the problem domain and to develop algorithms and data structure for different models for asynchronous algorithms for parallel and distributed system. We want scientific assumptions to make explain the concept for those models.
- Then, designing of the distributed systems with asynchronous processors has been developed (simulated) using graph theory, set theory properties of projective geometry and topology as basic tools for the explanation of asynchronous networks of processors.
- Finally the evaluation of the proposed algorithm in diverse topological has been chosen (simulated) for effective evaluation of the proposed work in asynchronous networks of processors for parallel and distributed system environment.
- Study of projective geometry and topology will be mapped to evaluate the obtained results related to the asynchronous networks.

- For this, the concept of polynomial time using polynomial number of processor for asynchronous system

IV. CONCLUSION

This paper has been surveyed a new applications areas of the distributed and parallel algorithms. Modern's computational world has many problems that's needed an urgent attention and consequently required an efficient solution for the same. Distributed algorithm for the Asynchronous networks processors is one of them which help to solve many problems like coloring of the graph etc. Next version of this article will present the algorithm for MPI system that's works on asynchronous architecture.

V. REFERENCES

[1] Pengfei Yang and Biao Chen "To Listen or Not: Distributed Detection with Asynchronous Transmissions", IEEE SIGNAL PROCESSING LETTERS, VOL. 22, NO. 5, MAY 2015.

[2] Ryan K. Williams, Andrea Gasparri, Attilio Priolo, and Gaurav S. Sukhatme "Evaluating Network Rigidity in Realistic Systems: Decentralization, Asynchronicity, and Parallelization", IEEE TRANSACTIONS ON ROBOTICS, VOL. 30, NO. 4, AUGUST 2014.

[3] Alexandre Maurer and Sebastien Tixeuil "Containing Byzantine Failures with Control Zones", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 26, NO. 2, FEBRUARY 2015.

[4] Haoyuan Li, Ali Ghodsi, Matei Zaharia, Scott Shenker and Ion Stoica "Reliable, Memory Speed Storage for Cluster Computing Frameworks", Technical Report No. UCB/EECS-2014-135, available at http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-135.html

[5] Michel Raynal "What Can be Computed in a Distributed System?",

[6] Matthieu Dreher, Bruno Ran "A Flexible Framework for Asynchronous In Situ and In Transit Analytics for Scientic Simulations", 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 2014, Chicago, United States. IEEE Computer Science Press. <hal-00941413>

[7] Ruiliang Zhang and James T. Kwok "Asynchronous Distributed ADMM for Consensus Optimization", Proceedings of the 31 st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

[8] S. G. Akl. The Design and Analysis of Parallel Algorithms. Prentice Hall, Englewood Cliffs, 1997.

[9] Bertsekas, D.P., and J.N. Tsitsiklis (1989). Parallel and Distributed Computation: Numerical Methods, Prentice Hall, Englewood Cliffs, NJ.

[10] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin,A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. Proceedings of the VLDB Endowment, 5(8):716–727, 2012.

[11] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pages 135–146. ACM, 2010.

[12] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of web-scale datasets. Proceedings of the VLDB Endowment, 3(1-2):330– 339, 2010.

[13] R. Power and J. Li. Piccolo: Building Fast, Distributed Programs with Partitioned Tables. In Proceedings of the 9th USENIX conference on Operating systems design and implementation, pages 293–306. USENIX Association, 2010.

[14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets: A FaultTolerant Abstraction for In-Memory Cluster Computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[15] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pages 423–438. ACM, 2013.

[16] Apache Oozie. http://incubator.apache.org/oozie/Luigi. https://github.com/spotify/luigi.

[17] Apache Crunch. http://crunch.apache.org/Apache Mahout. http://mahout.apache.org/

[18] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive a petabyte scale data warehouse using hadoop. In Data Engineering (ICDE), 2010 IEEE 26th International Conference on, pages 996–1005. IEEE, 2010.

[19] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In SIGMOD '08, pages 1099–1110.

[20] C. Chambers et al. FlumeJava: easy, efficient dataparallel pipelines. In PLDI 2010.

[21] Harel, D., Feldman, Y.: Algorithmics, the spirit of computing, 572 pages. Springer (2012).

[22] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (1978).
[23] Raynal, M.: Distributed algorithms for message-passing systems, 515 pages. Springer, ISBN:978-3-642-38122-5.
[24] Lamport, L.: On inter-process communications, Part I: Basic formalism. Distributed Computing 1(2), 77–85 (1986).
[25] Herlihy, M.P.: Wait-free synchronization. ACM Transactions on Programming Languages and Systems 13(1), 124–149 (1991).
[26] Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM 32(2), 374–382 (1985).
[27] Loui, M., Abu-Amara, H.: Memory requirements for agreement among unreliable asynchronous processes. Advances in Computing Research 4, 163–183 (1987).
[28] G. Zhao, J. R. Perilla, E. L. Yufenyuy, X. Meng, B. Chen, J. Ning, J. Ahn, A. M. Gronenborn, K. Schulten, and C. Aiken, "Mature HIV-1 Capsid Structure by Cryo-electron Microscopy and All-Atom Molecular Dynamics," pp. 643– 646, 2013.
[29] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma, "In situ visualization for large-scale combustion simulations," Computer Graphics and Applications, IEEE, vol. 30, no. 3, pp. 45–57, 2010.
[30] W. Gu, G. Eisenhauer, K. Schwan, and J. Vetter, "Falcon: On-line monitoring for steering parallel programs," in In Ninth International Conference on Parallel and Distributed Computing and Systems (PDCS'97), 1998, pp. 699–736.
[31] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in nternational Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press, 2012, pp. 49:1–49:9.
[32] Shalev-Shwartz, S., Singer, Y., and Srebro, N. Pegasos: Primal estimated sub-gradient solver for SVM. In Proceedings of the 24th International Conference on Machine Learning, pp. 807–814, 2007.
[33] Niu, F., Recht, B., R´ e, C., and Wright, S.J. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In Advances in Neural Information Processing Systems 24, 2011.
[34] Langford, J., Smola, A., and Zinkevich, M. Slow learners are fast. In Advances in Neural Information Processing Systems 22, 2009.
[35] Agarwal, A. and Duchi, J.C. Distributed delayed stochastic optimization. In Advances in Neural Information Processing Systems 24, 2011.
[36] Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J.K., Gibbons, P.B., Gibson, G.A., Ganger, G., and Xing, E. More effective distributed ML via a stale synchronous parallel parameter server. In Advances in Neural Information Processing Systems 26, pp. 1223–1231, 2013.
[37] Li, M., Andersen, D.G., and Smola, A. Distributed delayed proximal gradient methods. In NIPS Workshop on Optimization for Machine Learning, 2013.
[38] Bertsekas, D.P. and Tsitsiklis, J.N. Parallel and Distributed Computation. Prentice Hall, 1989.
[39] Zhu, H., Cano, A., and Giannakis, G.B. Distributed consensus-based demodulation: Algorithms and error analysis. IEEE Transactions on Wireless Communications, 9(6):2044–2054, 2010.
[40] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends in Machine Learning, 3(1):1–122, 2011.
[41] D. Chazan and W. Miranker, Chaotic relaxation, Linear Algebra Appl., 2 (1969), pp. 199–222.
[42] G. Baudet, Asynchronous iterative methods for multiprocessors, J. Assoc. Comput. Mach., 25 (1978), pp. 226{244. -46.
[43] Bertsekas, D.P. (1982). Distributed Dynamic Programming. IEEE Transactions on Automatic Control, AC-27,610~16.
[44] J. N. Tsitsiklis, On the stability of asynchronous iterative processes, Math. Systems Theory, 20 (1987), 137-153.
[45] B. Parhami and M. Rakov, "Perfect Difference Networks and Related Interconnection Structures for Parallel and Distributed Systems", IEEE Trans. on Parallel and Distributed Systems, vol. 16, no. 8,pp. 714-724, Aug. 2005.