Make-span Improvement with Data Partitioning for IoT Frameworks

Himadri Sekhar Paul
TATA Innovation Labs
TATA Consultancy Services Ltd
Kolkata, India
HimadriSekhar.Paul@tcs.com

Arijit Mukherjee
TATA Innovation Labs
TATA Consultancy Services Ltd
Kolkata, India
Mukherjee.Arijit@tcs.com

Swarnava Dey
TATA Innovation Labs
TATA Consultancy Services Ltd
Kolkata, India
Swarnava.Dey@tcs.com

ABSTRACT

With the current emphasis on intelligent infrastructures, sensor based ubiquitous intelligent systems, commonly known as 'Cyber-Physical Systems', have become important. Data acquisition, management, and analysis for knowledge extraction will give rise to a new generation of infrastructure and services encompassing every aspects of our daily lives. Such analysis are performed by well-known algorithms, having various constraints, including soft-real time constraints. This will create the need for a computing infrastructure where data parallel applications can be run. Data distribution for such applications assumes an important role for the performance of the system. In this paper, we address the problem of data partitioning as part of data distribution, such that parallel analysis of the partitions can minimize the overall run-time of the analysis. We investigate the problem under the scenarios where the communication links and computation nodes are unreliable and intermittently unavailable. We assume all such non-availabilities are known to the partitioner and propose an algorithm which generates a static partition of the data based on the capacity and availability of the elements in the system.

Keywords

Availability-constraint, Scheduling, Distributed system

1. INTRODUCTION

The next generation *cyber-physical systems* will be composed of networked infrastructures of smart objects. In such an infrastructure, objects will be able to *sense* the physical environment in which they belong through the sensory devices attached to it. Applications running on the such infrastructures will be able to *extract* sensory data, *analyze* them and generate control signals on which objects will *respond*. Such an infrastructure will transform the Internet into *Internet of Things* (IoT) where 'things' will interact over the communication framework provided by the Internet. Cyber-physical systems will bring opportunity to offer applications and services in multitude of domains including e-Governance, health-care, transport system, water services, energy utilization, waste management, and many more. Effective functioning of the services will depend on the precise analysis of the data accrued through the ubiquitous sensor devices. As an example, in the health-care domain, data-mining techniques can be used to create clusters or groups of persons with similar physiological conditions, which will offer the health-care professionals the opportunity to diagnose the condition based on knowledge gained from clustering. Real-time analysis of streaming data in a medical emergency unit may lead to early detection of impending medical conditions and alert the caregivers. Analysis of daily energy usage per appliance within home or office may form the basis of a predictive system for energy & utility systems.

An IoT platform is expected to manage analytical applications with diverse characteristics, such applications with periodic tasks, applications handling large data, applications having soft real-time constraints, etc. Distributed system is advocated to meet the diverse requirements of such applications. The computation infrastructure of an IoT platform is essentially a distributed system containing high end servers. Recently, it has been suggested that edge devices may also be included in the platform [5]. The sheer number of edge devices which is projected to come into use in the near future, makes such devices a lucrative source of computation power. In effect, tomorrow's IoT infrastructure will be geared to exploit power of a wide range of computing resources. However, applications (legacy or otherwise) which are not inherently distributed will not be able to effectively utilize the computing infrastructure. In many cases it is not practical to re-write the application to harness a distributed computing platform because the source code may not be available, or libraries used in such applications do not allow them to transform to the required platform, or simply the effort required for the transformation is not feasible. Such applications are treated as 'black boxes' and are executed without modification. In this class of black box applications, some may be data parallel applications, where the application can be executed in parallel on subsets of input data and the individual results can be combined to the desired result with comparative ease. This pertains to the class of problems addressed by the Map-Reduce technique [6]. However, this black-box execution proposition has a distinct advantage of treating the application as it is, without having to

ComNet-IoT'14 January 4-7, Coimbatore, India

re-write the same, which is in contrast to that in the Map-Reduce framework.

In this paper, we consider the class of applications which can be treated for data parallel black-box-style execution, where the input data can be partitioned into arbitrary smaller subsets and multiple instances of the same application can be executed, in parallel, on every data subsets. We assume the existence of a *combiner* application which can then gather the results from the execution instances and combine them into a unified result. A cloud computing platform in the IoT context is expected to contain heterogeneous machines, like sensor gateways, personal computers at home, or mobile communication devices, etc., connected via links with heterogeneous characteristics. These devices are not dedicated computing resources and are usually connected via unreliable communication links. The availability of both the computing resources and communication links are bursty, and even the computation elements may not be available with their full power. In such a platform the performance of analysis of a data-set will depend on the appropriate partitioning and distribution of the data-set considering the availability pattern of the resources. In this paper we attempt to address the problem of data partitioning for black-box kind of data analysis applications in the IoT context with the objective to minimize the overall execution time of an analytical application.

This paper is organized as follows. Section 2 presents some related background on data distribution and partition in distributed computing domain, along with the complexity of the problem. A formalism of the problem is shown in Section 3. A heuristic to obtain a partition of data based on capacity of the participant nodes is presented in Section 4. Simulation results are presented in Section 5. Finally Section 6 concludes this paper with directions to future work.

2. BACKGROUND

One of the potential factor of improving the performance of a distributed platform is the effect of movement of data for computation. Bent *et al* analyzed the problem of scheduling when data movement is considered along with computation [4]. Kosar *et al* propose a batch processing system which is aware of data volume [12].

Google's Map-Reduce framework [6] effectively exploits the power of distributed system for many data intensive problems. The platform was later adopted as the Hadoop open source project [1] by Yahoo. Researchers are working on the techniques of improving the performance of Hadoop-like Map-Reduce frameworks. Authors in [20] investigate the effect of data locality for Map-Reduce jobs in Hadoop system and propose a data-locality aware scheduler for Hadoop. The *Quincy* scheduler balances the priority of the task with locality of data [11]. In a Map-Reduce framework skewness of data distribution hinders system throughput, when some reduce-servers are loaded with more data to process. Authors in [9] present a data partition scheme to balance the data processing load on reduce-servers. Their approach is a greedy one where the least loaded server gets the largest partition.

A computing system can be viewed as a producer-consumer system, where application and data are producers of computing demand and the computing elements are consumers of such demands. The rate of consumption of computing demand by a computing resource can defined as the *capacity* of the resource. Capacity of computing elements in a system plays an important role in the problem of data partitioning. Capacity models are not easy to define since there are interdependencies among different elements of computation. Some detailed models of computing capacity can be found in [7, 15, 16]. Authors in [2] use a simple model of capacity computation, which is a linear combination CPU clock speed, memory size, bandwidth, etc, and they partition a data set according to the capacity ratio of the nodes. Then they show that this simple method of partitioning can improve the performance of some simple Map-Reduce programs.

In this paper we propose a model of data partitioner such that the overall computation span of a job is minimized. The next section discusses the complexity of the problem we are addressing.

2.1 Complexity of the Problem

In this paper we consider an application $\mathcal{A}$ which processes a data-set and generates a result. Mathematically, we consider the application as a function, defined as $\mathcal{A} : \mathcal{P}(\mathfrak{D}) \rightarrow \mathfrak{R}$, where the range $\mathfrak{R}$ defines the set from which the return value of the algorithm $\mathcal{A}$ is drawn. The domain of the application is a set of data drawn from $\mathfrak{D}$, denoted as the power set of $\mathfrak{D}$, $\mathcal{P}(\mathfrak{D})$.

Let $\mathbf{D}$ be a member of $\mathcal{P}(\mathfrak{D})$ acting as an input set. Let $\mathbf{D} = \{d_1, d_2, d_3, \ldots, d_m\}$. We partition $\mathbf{D}$ into n segments $D_1, D_2, D_3, \ldots, D_n$ such that $\bigcup_{i=1}^{n} D_i = D$ and $D_i \cap D_j = \varnothing \ : \ 1 \leq i, j \leq n, i \neq j$ and typically $n \leq m$. In a distributed computing platform, n instances of $\mathcal{A}$ can be initiated with different partitions of the input set, *i.e.* , we have a set of tasks $\mathcal{J} = \{\mathcal{A}(D_1), \mathcal{A}(D_2), \ldots, \mathcal{A}(D_n)\}$. We assume that the existence of a combiner/reducer which takes care of the return values of these instances, but is outside the scope of this paper.

Completion time of each task $\mathcal{A}(D_i)$ will depend on various system and application parameters. We want to achieve an effective partition such that the *make-span* of $\mathcal{J}$, which in the field of research in scheduling refers to the latest completion time of the tasks, is minimized.

The problem of data partitioning can be reduced into a more generic and well researched field of scheduling. Instead of creating n partitions of $\mathbf{D}$, one can create m partitions, each containing exactly one, atomic data element. Execution of these m partitions with m instances of $\mathcal{A}$ gives us a job of m tasks $\mathcal{J}' = \{\mathcal{A}(d_1), \mathcal{A}(d_2), \ldots, \mathcal{A}(d_m)\}$, which now need to be scheduled on n resources.

However, in the context of IoT framework, nodes and communication links may not always be available. Scheduling under such availability constraint in multi-processor system has been studied [17,19]. The version of the scheduling problem we address in this paper falls in the class $Q|pmtn, rs|C_{max}$, implying that jobs can be *preempted* and *resumed* later, and processes are unavailable in multiple time-durations during the whole scheduling time-span, where the objective is to minimize the make-span of the jobs. The problem without any availability constraint ($P||C_{max}$) is known to be NP-hard, the problem with availability constraints ($P|rs|C_{max}$), where jobs can be resumed, is also hard [14]. Therefore, the non-uniform processing version of the problem $Q|rs|C_{max}$ is also hard. In our present work we address the problem of scheduling when the machines are of different speeds and the tasks are independent.

The problem of scheduling under availability-constraint is well studied. This section provides some relevant results for the class of scheduling problems we are addressing in this paper. Schmidt studied the problem $P_m|pmtn, rs|C_{max}$ and gave conditions under which a feasible preemptive schedule exists [18]. He shows that such a feasible solution can be constructed in $O(n + m \times log\ m)$, where m is number of machines and n is the number of jobs. He also showed that number of scheduler induced preemption is proportional to the number of processing intervals. Lawler and Martel proposed a pseudo-polynomial time algorithm for the scheduling problem involving two machines with preemptive jobs under total weighted tardiness minimization criterion, *i.e.* $Q2|pmtn, rs|\sum w_jU_j$ [13]. For more than two machines under the criterion of minimization of maximum lateness of jobs ($Q|r_j, pmtn|L_{max}$), a polynomial time algorithm was proposed by B̃lażewicz *et al* [3]. Gharbi *et al* present heuristics for problems in $P, NC_{inc}||C_{max}$ and $P|r_j, q_j|C_{max}$ and showed that their algorithm can work for large instances of the problems [8]. Hashemian in his masters dissertation [10] presents ILP formulations for some scheduling problem under availability constraints.

In this paper, we assume unavailability are advertised by the nodes or links and are, therefore, known in advance. We consider developing a static schedule and partition of data elements which takes into account the advertised availabilities. In the next section we present a heuristic to determine a schedule to reduce make-span of the job.

3. PROBLEM DEFINITION

We assume a distributed computing environment where one central node, acting as data partitioner (DP), is connected to n computing nodes, $P = \{P_1, P_2, \ldots, P_n\}$, in a star topology. The edges are associated with a weight, l_i, depicting link speed. A node is characterized by a tuple $\{s_i, v_i\}$, where s_i denote the CPU speed (clock speed) and v_i is the advertised non-availability of the node.

The data partitioner has in its possession the data $\mathbf{D} = \{d_1, d_2, \ldots d_m\}$. We assume the non-availability of the nodes and links are advertised. Under these constraints the problem is to achieve a partition of the data $\mathbf{D}$ into n partitions $D_1, D_2, \ldots, D_n$ and each partition D_i is assigned to process P_i, such that the make-span of the job is minimized. We say a partition of $\mathbf{D}$ is consistent iff,

$$\bigcup_{i=1}^{n} D_i = \mathbf{D} \quad (1)$$

$$D_i \cap D_j = \varnothing \quad \ldots \quad 1 \leq i, j \leq n, \ i \neq j \quad (2)$$

The model of the problem assumes the tasks to be homogeneous and independent, and therefore independent. This essentially implies that the data partition problem can be transformed into partition problem of the number of data elements.

4. HEURISTIC FOR MAKE-SPAN MINIMIZATION

In this section we present a heuristic to partition a data set $\mathbf{D}$ to assign to a set of n processes. The heuristic is based on the ratio-based data partition model, but it continuously refines the partitions considering advertised non-availability of the processes. The outline of the algorithm is presented as Algorithm 1.

The algorithm starts with a partition obtained by applying ratio-based partition from the capacity of the processes. This partition method considers that processes are always available. The make-span obtained from this partition is a *base make-span*, ms_e which does not consider intermittent unavailability and hence signifies a lower limit of the make-span for the given data set. Due to intermittent unavailability of a process, the completion time of processing the data partition assigned to it will be pushed away from its *base* completion time. This may effectively cause an elongation of the make-span of processing $\mathbf{D}$. The algorithm, then, tries to refine the partition to reduce the effective make-span by re-assigning some data element to other processes in the following manner. We define *slack time* as the duration $[ms_e, ms]$. It estimates total number of data elements processed during the slack time period and these are then redistributed using the ratio model. This iteration continues until there is no substantial improvement in the effective make-span in subsequent passes.

In the presentation of this heuristic, we assume the following model of capacity of a computing element, which consists of the communication link with the DP and the computing element itself. We define capacity as a linear composition of the cost of transfer of one data element form DP and processing cost of the element at the node. The *communication cost* is the estimated latency of transferring one data element from the DP to the node and can be expressed as a function of the link speed. The *computation cost* is the estimated time to process one data element in the node. The computation cost model is subject of independent research. Here we assume that some estimate of the cost can be obtained from some model involving the algorithmic complexity of problem, CPU speed, memory capacity at all levels, disk speed, etc [2]. In the heuristic presented in Algorithm 1, has to its disposal the tuple $< \mathcal{N}, \mathcal{P} >$ as the system model, where

- $\mathcal{N} : \mathbb{N} \rightarrow \mathbb{R}$: defines the network latency to transfer one data element from DP. $\mathcal{N}(i) = t_n$ implies that t_n time unit is required to transfer one data element from DP to P_i.
- $\mathcal{P} : \mathbb{N} \rightarrow \mathbb{R}$: defines processing speed of one data element. $\mathcal{P}(i) = t_p$ implies P_i requires t_p time units to process one data element.

Similarly the availability of each of the i^{th} resource elements can be described as $< \mathcal{V}_i >$, where,

- $\mathcal{V}_i : \mathbb{N} \rightarrow \{0, 1\}$ is a binary function which denotes whether link or computation node of a resource element is available at certain time unit and is defined as,

$$\mathcal{V}_i(t) = \begin{cases} true \ \ldots & P_i \text{ and link between DP and } P_i \text{are available at the } t^{th} \text{ time-unit} \\ false \ \ldots & \text{otherwise} \end{cases}$$

We define the following vector of functions to characterize availability of computation elements in the system.

$$\vec{\mathcal{V}} =< \mathcal{V}_1, \mathcal{V}_2, \ldots \mathcal{V}_n >$$

We assume time units to be discrete, *i.e.* if time unit is in seconds, then $\mathcal{V}_i(t) = 0$ implies the i^{th} resource element is unavailable for the full t^{th} second.

4.1 Proof of Correctness

The proof of correctness of the heuristic involves proving

- *Consistency of the partition.* The data partition obtained from the heuristic is consistent, *i.e.* it satisfies the conditions of Equations 1 and 2.
- *Termination.* The heuristic terminates in finite time.

Our heuristic is based on the ratio-based partition and we assume that the ratio-based partition satisfies all the above correctness criterion. Based on this we prove the followings.

PROOF. *Consistency of Partition:* The proof of the consistency rests on the ratio-partition algorithm, which serves as the basis of our heuristic. We assume that the partition of $\mathbf{D}$ obtained from ratio-model at the line 1 of Algorithm 1 is consistent. So at this line the invariant $\sum_{i=1}^{n} D_i = \mathbf{D}$ is satisfied.

The for-loop starting at line 1 calculates the aggregate number of data elements, ΔD, which will be processed in the *slack time*, *i.e.* beyond the ideal make-span time-line (ideal make-span is computed without considering unavailability of the processes). At line **??** the algorithm determines the count of data elements processes in the slack time for i^{th} process as Δd. The Δd is cumulated in ΔD at line 1 before being subtracted from the D_i at line 1. Therefore at the end of the for-loop at line 1 the invariant $\left(\sum_{i=1}^{n} D_i\right) + \Delta D = \mathbf{D}$ holds.

The ratio-based partition at line 1 partitions only ΔD and by the consistency guarantee of the algorithm the invariant $\sum_{i=1}^{n} \Delta D_i = \Delta D$ holds. Therefore, the invariant $\left(\sum_{i=1}^{n} D_i\right) + \left(\sum_{i=1}^{n} \Delta D_i\right) = \mathbf{D}$ also holds.

Finally the values of ΔD_i's are added with the $D_i's$ in the for-loop at line 1. So, at the end of the for-loop, invariant $\sum_{i=1}^{n} D_i = \mathbf{D}$ holds. Therefore at the end of the algorithm the condition $\sum_{i=1}^{n} D_i = \mathbf{D}$ holds and D_i's can be used to obtain a consistent partition of $\mathbf{D}$. Hence proved. □

Now we prove the termination property of the algorithm.

PROOF. *Termination:* First we show that the Algorithm 2 terminates. To show this we need to show that condition at line 2 is satisfied. The initial value of $last_\tau$ is the completion time of analysis of $\mathbf{D}$ data elements. In the repeat-until loop of the algorithm, the value of τ monotonically increases. Therefore, the value of c_τ also monotonically increases. Since the the value of c_τ is the count of unavailable slots in τ duration and there are finite number of unavailable slots, increment of c_τ is also finite. Therefore, the condition at line 2 will be satisfied in finite time.

Now we show that the unconditional loop encompassing lines 1 - 1 of algorithm 1 is terminated by execution of break at line 1 in finite time. We need to show that the if-condition checking at the line 1 is successful in finite execution time.

The if-condition at line 1 checks the difference between the make-span of the previous run of the loop with the make-span corresponding to the refined partition of $\mathbf{D}$. In each iteration of the loop the make-span value computed in ms should steadily decrease, otherwise the if-condition is satisfied and the algorithm terminates.

Algorithm 1: Makespan-Refinement

input : $\mathbf{D}, \mathcal{N}, \mathcal{P}, \vec{\mathcal{V}}$
output: A partition of the data
begin
  Partition $\mathbf{D}$ as $(D_1, D_2, D_3, \ldots D_n)$ using ratio-based model ;
  for $1 \leq i \leq n$ **do**
    /* Compute completion time of each partitions without availability information */
    $\mathcal{C}_i = CompletionTime\,(D_i, \mathcal{N}(i), \mathcal{P}(i), \varnothing)$;
  /* Compute make-span without considering unavailability of the processes */
  $ms_e \leftarrow max\,\{\mathcal{C}_i : 1 \leq i \leq n\}$;
  $last_{ms} \leftarrow ms_e$;
  repeat
    for $1 \leq i \leq n$ **do**
      /* Compute completion time of each partitions using availability information */
      $\mathcal{C}_i \leftarrow$
      $CompletionTime\left(D_i, \mathcal{N}(i), \mathcal{P}(i), \vec{\mathcal{V}} \uparrow_i\right)$;
    /* Compute make-span with unavailability information */
    $ms \leftarrow max\,\{\mathcal{C}_i : 1 \leq i \leq n\}$;
    if $last_{ms} \leq ms$ **then**
      /* No more refinement of partitions possible */
      break from loop
    $\Delta D \leftarrow 0$;
    for $1 \leq i \leq n$ **do**
      /* *Slack time* : computation effort of P_i on D_i data elements in the interval $[ms_e, ms]$ */
      $\Delta d \leftarrow \lfloor \frac{ms_e - ms}{\mathcal{N}(i) + \mathcal{P}(i)} \rfloor$;
      if $\Delta d > 0$ **then**
        /* Calculate the number of data elements processed in the slack time */
        $D_i \leftarrow D_i - \Delta d$;
        $\Delta D \leftarrow \Delta D + \Delta d$;
    Partition ΔD as $(\Delta D_1, \Delta D_2, \ldots \Delta D_n)$ using ratio-based model. ;
    for $1 \leq i \leq n$ **do** $D_i \leftarrow D_i + \Delta D_i$;
    $last_{ms} \leftarrow ms$;
  until;
  Return partition $(D_1, D_2, \ldots D_n)$;

Algorithm 2: CompletionTime

input : $\mathbf{D}, l, c, \mathcal{V}$
output: Completion Time
begin
[Compute completion time for $\mathbf{D}$ *data elements on a resource element with communication cost as* l *and computation cost as* c*, where the availability information (per time unit) for the computing elements is described by functions* $\mathcal{V}$*'s]*
$last_\tau \leftarrow \tau \leftarrow \mathcal{T} \leftarrow (|\mathbf{D}| \times (l + c))$;
repeat
  $c_\tau \leftarrow 0$;
  for $1 \leq t \leq \tau$ **do**
    if $\mathcal{V}(t) = false$ **then**
      /* t^{th} time is unavailable */
      $c_\tau \leftarrow c_\tau + 1$;
  $\tau \leftarrow \mathcal{T} + c_\tau$;
  if $last_\tau = \tau$ **then** *break from loop;*
  $last_\tau \leftarrow \tau$;
until;
return τ ;

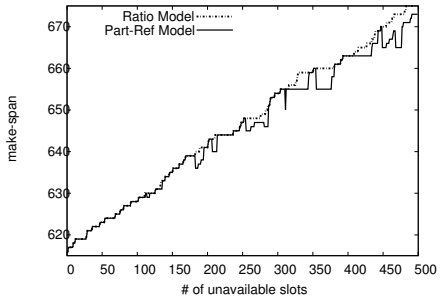
According to the algorithm, the value of ms_e signifies the make-span for processing the data-set $\mathbf{D}$ with n processes, without considering the unavailable time periods. It signifies an ideal case and there is no possible partition with make-span less than ms_e. However, due to intermittent unavailability of processes, completion time for individual process increases (refer Algo 2), and as a result the effective make-span (ms) may be more than ms_e, *i.e.* $ms \geq ms_e$. The difference $ms - ms_e$ is upper bounded by T_A, the maximum of the unavailable periods for all processes in the interval $[0, ms]$. Since $ms \geq ms_e$, $last_{ms} \geq ms_e$. Therefore, $(last_{ms} - ms) < T_A$.

Since ms must monotonically decrease with each iteration of the loop, *i.e.* after each iteration $last_{ms} > ms$ and $(last_{ms} - ms) < T_A$, where T_A is finite, the loop must terminate after finite number of iterations. □

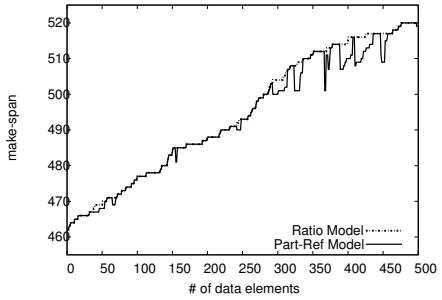
5. EXPERIMENTS AND RESULTS

The experiments were conducted on a simulation framework and the make-spans of the tasks were recorded. The framework creates static partition of a given number of data. We used ratio-based partitioning to compare against the partition created by our heuristic. Our experiment set-up includes 10 nodes and simulated under 3 different scenarios depicted below.

- *Scenario #1*: A set of homogeneous nodes (*i.e.* with almost same computation power) are connected to the data partitioner (DP). The link between the DP and the nodes are homogeneous (*i.e.* with almost same link speed).
- *Scenario #2*: We simulate two clusters of nodes, C_1 and C_2. The nodes in individual clusters are homogeneous. Nodes in cluster C_1 are of high computation speed and those in cluster C_2 are of lower computation speed. The link speed between the DP and nodes in C_1 are of higher than that between the data partitioner and nodes in C_2.



(a) For increasing unavailability (Scenario#3)



(b) For different data volumes (Scenario#2)

Figure 1: Comparison of Make-Spans

- *Scenario #3*: This scenario is similar to the scenario#2, except that the link speed between the DP and nodes in C_1 are of lower than that between the DP and nodes in C_2.

5.1 Variability of Unavailable Slots

The number of unavailable slots were varied with random assignment of unavailable slots to processes and the make-span of the schedule is observed. Figure 1(a) shows comparison of make-spans for scenario #3, where high-speed machines are 4 times faster than the low-speed machines, high-speed links are twice as fast as the low-speed links, and high-speed machines are connected on high-speed links and low-speed machines are connected by low-speed links. The horizontal axis shows total number of unavailable slots which are randomly distributed among processes.

The result shows that with the increase of number of unavailable slots the make-span obtained from ratio-based partition increases. This is expected since the ratio-based partition does not consider unavailable slots. Our algorithm tries to redistribute work to obtain a balance of work loads based on the capacity of the elements with the objective to reduce make-span. As the number of unavailable slots increases, there is more scope to perform refinement and improve make-span. In this case, the factor by which a computation is done by high speed machines is higher than the factor by which a high-speed link transfers data. In this condition, when a low-speed machine is unavailable for a duration, the job can be transferred to a high-speed machine with effective reduction in make-span. Our method generates data partition and distribution such that the make-span is better in most of the cases for all the scenarios.

5.2 Variability of Data Volume

We varied the number of data elements and compared the make-span of the job, given that a total of 500 time units are unavailable across all processors. Figure 1(b) shows the comparison the make span for the scenario#2, where the high speed machines are 4-times as fast as the low-speed machines, link speed of a high-speed connection is 2-times faster than a low-speed connections, and high-speed machines are connected to low-speed links and low-speed machines are connected to high-speed links. The results of other cases are similar and not presented here. In these cases our heuristic could exploit the unavailable information and produce a make-span which is at-least comparable to ratio-based model and out performs in many cases.

6. CONCLUSION AND FUTURE WORK

An IoT framework is a conglomeration of heterogeneous resources connected by heterogeneous communication. Incorporating these resources in a framework, where their computation powers can be effectively harnessed, poses several challenges. In this paper we address the problem of data partitioning for data parallel applications, under the scenario where nodes may be unavailable spuriously in a day. We present a formulation of the problem and present a heuristic for the problem.

The model we present in this paper assumes the availability of the nodes are advertised and we exploit this knowledge to generate a static schedule. However, in practical scenarios all unavailability may not be advertised. Statistical model of availability can be incorporated in the model to address non-advertised unavailability, like transient faults.

7. REFERENCES

[1] Hadoop. `http://hadoop.apache.org`.

[2] R. Arasanal and D. Rumani. Improving MapReduce performance through complexity and performance based data placement in heterogeneous hadoop clusters. In *Intl. Conf. on Distributed Computing and Internet Technology (ICDCIT)*, feb 2013.

[3] J. Bl̃ażewicz, M. Drozdowski, P. Formanowicz, W. Kubiak, and G. Schmidt. Scheduling preemptable tasks on parallel processors with limited availability. *Parallel Computing*, 26(9):1195 – 1211, 2000.

[4] J. Bent, D. Rotem, A. Romosan, and A. Shoshani. Coordination of data movement with computation scheduling on a cluster. In *Challenges of Large Applications in Distributed Environments, 2005. CLADE 2005. Proceedings*, pages 25–34, 2005.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proc. of the 1st Mobile Cloud Computing*, MCC-2012, pages 13–16. ACM, 2012.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[7] A. Fionov, Y. Polyakov, and B. Ryabko. Application of computer capacity to evaluation of Intel x86 processors. In F. L. G. et al, editor, *Proc of the 2nd Intl Congress CACS, IASC*, pages 99–104, 2012.

[8] A. Gharbia and M. Haouari. Optimal parallel machines scheduling with availability constraints. *Discrete Applied Mathematics*, 148:63–87, 2005.

[9] B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Handling data skew in MapReduce. In *CLOSER*, pages 574 – 583, 2011.

[10] N. Hashemian. Makespan minimization for parallel machines scheduling with availability constraints. Master's thesis, Dept. of Industrial Engg., Dalhousie University, Halifax, Nova Scotia, 2010.

[11] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy : Fair scheduling for distributed computing clusters. In *ACM SIGOPS 22nd Symp. on Operating Systems Principles*, SOSP'09, pages 261 – 276. ACM, 2009.

[12] T. Kosar and M. Balman. A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computer Systems*, 25(4):406 – 413, 2009.

[13] E. L. Lawler and C. U. Martel. Preemptive scheduling of two uniform machines to minimize the number of late jobs. *Operations Research*, 37:314–318, 1989.

[14] C. Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9:395–416, 1996.

[15] A. Rakitskiy, B. Ryabko, and A. Fionov. Evaluation of computer capacity for P5 Intel processors. In *Problems of Redundancy in Information and Control Systems (RED), XIII International Symposium on*, pages 70 –73, sept. 2012.

[16] B. Ryabko. Using information theory to study efficiency and capacity of computers and similar devices. *Information*, 1(1):3–12, 2010.

[17] H. R. D. Saidy and M. T. Taghavi-Fard. Study of scheduling problems with machine availability constraint. *Journal of Industrial and Systems Engineering*, 1(4):360–383, 2008.

[18] G. Schmidt. Scheduling on semi-identical processors. *Zeitschrift fÃijr Operations Research*, A28:153–162, 1984.

[19] G. Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121:1–15, 2000.

[20] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin. Improving MapReduce performance through data placement in heterogeneous hadoop clusters. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–9, 2010.