Algorithm for CPU resource allocator case study and comparing between ordinary and ML Algorithms

Azeez Ammar Emad & Bondarenko Yulia Valentinovna

Voronezh state university, Russia  
ammar.azeez548@gmail.com

Abstract
The problem of resource allocation of scarce is a very important task especially in dynamic and non-stable environments when the importance and priority of recourse are changing while the system is already working. In our work, firstly, we simulate the problem by mathematical model than the essential part is to use intelligent algorithms like comparing random scheduling, hill climbing, simulating annealing and genetic algorithms, by using these tactics of such kind of algorithms we designed an algorithm that can handle with every type of multi-agent learning-based resource allocation for distributed systems that act place of most promising approaches to these kinds of allocating systems problems. The approach will be authorized by achieving on different types of large scale systems than correlated with the regular algorithms to compare the result and get the efficacy of new algorithm.

Keyword: resource allocation, intelligent algorithm, memory allocator, CPU deadline, machine learning.

Introduction:

There are several types of algorithms that have been developed to solve the problems of resource allocator but according to the increasing of a domain of environment for such kind of problems we need to design intelligent algorithms, for example, the development of communication technology made the request for the frequency band is very high, so the servers must find the best, suitable, easy to use and efficient in multi-agent form for dynamic systems [1]. As well as the case of complex economic systems, scheduling of CPU working, manufacturing process production control, mobile network and so on there are many of large-scale systems can be achieved and work base on the resource allocator smart algorithms.
In general, all systems in the real-world can be classified under the distributed resource allocation and related to combinatorial optimization problems. Such as massively parallel (GRID) [2] that control of a finite set of a reusable scarce resource to reallocating for known temporarily extensions. however, most of the classical algorithms concentrate on fixed and deterministic situations, by contrast, real-world situations when the environment is unsure and change dynamically, so this complexity the classical solution applicability. Even several new algorithms had appeared to handle the problems of large-scale, unfixed and dynamic environment [5], but it still not speed up enough for the computation and more robust solution.

The phrase of recourse allocator itself refer to the control of recourse so in this point we identify in each case study the main elements in environment under the studying such as storage capacity, cloud resources, network frequency and all the element of others field specially that dynamically changes [10]. In this paper we formulated as a brief survey of deferent researches and various types of resource allocating environments that try to get the ideal using of CPU resource which consider to reach the best management include [9], and get the memory requirements to maintain the processing time for all tasks, , and which search for an optimal reallocate in a finite space of possible recourses thus, they do not try all the possible solutions but reduce the time and use clever or smart algorithms and machine learning to know already the ideal recourse allocator for every task so in our work we will display two different algorithms in the same environment in our case the environment of CPU and compare and evaluate the way of works and give solutions of the weak pints.

**Architecture of the environment**

In this section we will discuss the recourse of CPU that consist of the memory and all capacity of proceeding and do the I/O operations. The schedule of processing system of CPU must provide the minimum quantity of required assets to execute the activates without reach the deadline, even if there are extra sources available, therefore the scheduler of the sources to do the activity. This way jobs potentially make more development, For example, CPU-bounded jobs will make greater progress as greater CPU they have allotted. However, a more allotment of memory does not mean a fast of execution time if the venture operating set already suits in reminiscence, growing the allotted memory to the assignment will no longer make the its execution time faster. Thus, the provider gets in advance with the work, by means of fully the usage of the assets while the system is closed to deadline so that you
can have extra resources to be had for future overload periods. According to virtualization, changing dynamically [3]. Once a task is completed, the Scheduler stores the activity’s execution information—process, useful resource allotment and execution time—in the Prediction System for use to train the SAP. In the other hand there are the multicores processors that contain a large-scale souses that must be controlled in some how to get the maximum benefits of such kind of CPUs, there are two ways to gain better performance for multicores machines by increasing the wide variety of CPUs, and growing the quantity of vector devices in one SIMD processor. Or by design really scalable algorithm ought to take advantage of both. However, most past research on scalable memory allocators scales well with the quantity of CPUs, this will lead the research to study the all resource available and in the case of CPU we have the GPU that consider as important processes resource that will increase the speed and efficient of proceeding operation but in this case must use the parallel algorithms. However, dynamic memory allocation isn't always to be had in parallel code in many-center programming models including CUDA and Open CL [4]. This poses a barrier to supporting higher-level programming languages such as C++ that rely more closely on memory allocation. Most many-middle GPU architectures run parallel threads simultaneously on SIMD processors to achieve higher computing throughput. SIMD designs reduce hardware price by means of exploiting information parallelism. However, SIMD-parallel operations that get entry to the identical resource ought to nevertheless be serialized. A lock-loose algorithm will certainly serialize itself but may induce intense slowdown. We observed this effect in a simple model of XMalloc. Our designed allocation algorithm avoids this effect by way of combining memory requests, using one thread to allocate on behalf of the complete SIMD processor [11].

**Display Algorithm and formulation of problem:**

Different CPU recourse allocator and scheduling algorithms have extraordinary properties, and the choice of a selected algorithm may favor one elegance of procedures over another. In selecting which algorithm to use in a particular situation, we must consider the residences of the numerous algorithms [8]. Many criteria have been counselled for comparing CPU scheduling algorithms. Which traits are used for the assessment can make a tremendous distinction in which algorithm has chosen to be best. The criteria consist of the following:

1. Utilization/Efficiency: keep the CPU busy
100% of the time with beneficial work.

2. Throughput: reduce the variety of jobs processed per hour.

3. Turnaround time: from the time of start the task to the time of its done, reduce the time batch users ought to wait for the output.

4. Waiting time: Sum of times spent in the equipped queue - reduce this.

5. Response Time: time from submission task until the first reaction is produced, minimize reaction time for the users.

6. Fairness: ensure each method receives a fair percentage of the CPU.

**PREDICTION SYSTEM**

The Prediction System shops and manages information about previous process executions that allow you to predict the amount of required resources –CPU percentage and reminiscence allocation– by an activity to be finished earlier than its deadline. This prediction is done by two components: the AP and the SAP.

A. Analytical Predictor (AP). The intention of the AP is to be expecting process aid requirements while the SAP isn't trained. As regards the prediction of CPU requirements, we get on for analytically \( T_{cpu} \), which is the processing time of a CPU bounded job considering that it has allocated a CPU share of \( \%CPU \), by the following equation: 

\[
T_{cpu} = \frac{T_{ref}}{\%CPU} \tag{30}
\]

where \( T_{ref} \) is refer to the execution time of the job weighted by the CPU share that was allocated to that execution \( e \) \( (T_{ref} = T_{cpu, e\%CPUe}) \). This data is turned from the most similar –in terms of processing time– past execution of the same tasks with the same input data size.

So, the AP module can consist of the minimum CPU requirements as follows:

\[
\%CPU = \frac{T_{ref}}{Deadline - Now} \tag{30}
\]

where Deadline is the deadline for completion of the task, and Now is the time at task working. And this will lead to whenever the Deadline is in the future \( (Deadline > Now) \), \( \%CPU \) is a positive value. Then, the deadline possibly can be accomplished if \( \%CPU \leq 100 \).

**Algorithm improved**

To get more efficient and fast result and to reach the optimal recourse allocating we can use the Self-Adjusting Predictor (SAP) that depend on the machine
learning (ML) to build a smart algorithm and this will provide a flexibility in the
CPU behavior and also for the memory. The SAP has the functionality to study
from unique tasks and expect the sources wished from completely one-of-a-kind
tasks the use of a single model. Additionally, new activity executions are
constantly used to build new models, which replace the used ones whilst higher
exactness is achieved. We have chosen a few ML algorithms to build the models
that perform the predictions: Linear Regression, M5P [6], REPTree and Bagging
[7]. Basically, due to the fact, their prediction accuracy is excessive and their
solving complexity is low—introducing a low overhead, which is important in an
on-line system. And only through the experimentation, we can choose which
algorithm is more appropriate to predict each resource. To allocate the recourse of
CPU or memory or both by using ML there are several strategies as flowing:

1) Predicting CPU: by using the linear relationship between the submissions the
tasks to the CPU and execution time we can provide a data the can the ML use it for

The certainty of these fashions in terms of mean absolute errors (MAE) is proven
in Table I. M5P is the set of rules that performs higher predictions on average –
2.35 units of CPU per cent in absolute phrases, or 5% of relative error– followed
via Bagging with M5P [12].

| TABLE I |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| MAE for (UNITS OF %CPU) PREDICTING CPU |
| LR | M5P | REP tree | Bagging for M5P | Bagging for REP tree |
| 10.78 | 2.35 | 5.04 | 3.04 | 4.18 |

2) Predicting CPU according to input size: here we recall the same environment
but now thinking about that the tasks have unique enter facts sizes. For example we had
a 5 sizes for training and 4 sizes for testing process. In this case, we also have to do
a transformation to the example but in this case we take in mind the
arguments (input records, size and execution time).

So the correlative execution time $\frac{\text{size of data}}{\text{execution time}}$ [12], in table II we can see the
MAEs of the fashions constructed using the ML algorithms. We can study that
Bagging with M5P is the best algorithm to make this sort of predictions −11% of
relative error. If the expected capacity of CPU is greater than the required one, we
are over-provisioning the tasks with resources, though the task would likely use this CPU surplus to finish its execution earlier. In this way, the greater the quantity of CPU to predict the worst predictions. The know-how of the error tendency could assist us in designing scheduling guidelines that cope with these errors.

### TABLE II

**MAE (UNITS OF %CPU) PREDICTING CPU TAKING INTO ACCOUNT INPUT DATA SIZE.**

| Application | LR   | M5P  | REPTree | Bagging M5P | Bagging REPTree |
|-------------|------|------|---------|-------------|-----------------|
| all         | 15.39| 8.69 | 8.92    | 6.77        | 7.11            |

3) Predicting memory: To look at how well the amount of memory required by the task can be predicted, we use the same packages as inside the case of CPU plus a memory extensive micro benchmark. We designed and test the design with real executions using numerous memory allotments. The behavior of the tasks is the following: if allocated with the required reminiscence or more the execution time does no longer vary—most effective very lightly if the application is implemented in Java due to the fact, the resource more frequently with lower allotments—so, the reminiscence allotment and the execution time isn't correlated, making it impossible to predict accurately.

### TABLE III

**MAE for (MB) PREDICTING MEMORY**

| Application | LR   | M5P  | REPTree | Bagging M5P | Bagging REPTree |
|-------------|------|------|---------|-------------|-----------------|
| all         | 190.76| 200.45| 178.06  | 179.45      | 158.35          |

All previous algorithm and strategies did not work with the multicores CPUs so on the other hand a shared-memory multi-middle or many-center machines, many processing gadgets percentage the shape of the same records. To make certain the consistency of these concurrent recourses, there is want to synchronize their accesses. In such a device the programmers generally ought to use synchronization primitives along with semaphores, monitors guarded statements, and mutex locks, but algorithms based on locks do now not scale well. Consequently the operations
of different processes on the shape of a shared record should appear like serialized: if two operations execute simultaneously, the gadget ensures the equal end result as if one of them is arbitrarily executed before the other [5].

**ACHIEVING SIMD SCALAB:**

Most many-middle GPU architectures work by parallel threads concurrently on SIMD processors to get the higher computing throughput. SIMD working to get lessen hardware fee by means of exploiting information parallelism. Although, SIMD-parallel programming that get right of entry to the same resource has to still be serialized in somehow. A lock-free set of rules will naturally serialize itself, however, may induce severe slowdown. We found this impact in a simple version of XMalloc. Our scalable allocation set of rules avoids this impact by using combining memory requests, the usage of one thread to allocate on behalf of the complete SIMD processor [11].

Display the XMALLOC’S DESIGN:

The XMalloc algorithm for allocating memory is designed to grant closely multithreaded tasks with high-throughput access to a shared memory pool. Parallel allocators commonly use thread-private facts systems to get the best scalability [5]. So this kind of memory allocator deals with the number of processors and the number of memory units in a processor. We design a solution to transform traditional CAS (compare-and-swap) based lock-free algorithms to deal with multi SIMD processing unit. In XMalloc, this solution is related with aggregate SIMD-parallel memory allocation.

**The use of Atomic CAS-based Lock-free Algorithms**

SIMD execution of CAS-based complex sections produces efficient performance. Because all SIMD threads on one CPU working and executing in lockstep, they execute multi-section simultaneously.

An $N$-wide SIMD CPU will get in to the loop by the critical of section at least $N$ of times, so as a result if all SIMD threads work serially. This will make the process slower, because multiple SIMD threads are run in each iteration, the shared tasks and values will be accessed by CAS statements at least $(N+1)*N$ 2 times [11], having more memory traffic than ordinary serial execution.
The idea behind our unsalable malloc function is to make the list of memory free and this will lead to find critical of memory section that re-executed as free memory and this will provide at least on free block of memory at a time which is extracted from the free list. Figure 1 illustrates how algorithm is transformed to take away this problem, using four SIMD-parallel memory reallocate. The algorithm first cooperatively sum their asked memory sizes using a previous sum scan. Then, one thread gets a memory place of this size.

So the main idea here is to use the unused memory blocks that can be reached by using the XMalloc’s allocation algorithm.

**Discussion the result and comparing algorithms**

In the first case the ML techniques in a Self-Adjusting was used to produce specified sources to fulfil a given service-degree metric the usage of the outcomes from previous executions. As the rearrange the CPU capacity, the algorithm achieve high prediction accuracy (11% of relative mistakes) using the Bagging with M5P algorithm. Regarding memory prediction, a 17% of relative error is achieved the usage of Bagging with REPTree. Moreover, this characteristic, along
with the low overhead incurred, makes the algorithm gadget appropriate for making selections online. Besides, it’s proposed an Analytical. But this algorithm didn’t discuss the multicore CPUs, on the other hand for recourse the multicore processes and their memory used other strategy and algorithm, XMalloc algorithm and execution of CAS-based complex sections as a result of the work the algorithm could use all the free value of memory and capacity of processers but in this algorithm we could not use the AI that make se easy to use in the first case to forecasting about the recourse that the task need before submission the task to processer and this attributer made the execution very fast and more efficient.

Conclusion and future work

Allocate problem with a single resource constraint was direct but with the environment is up to date continuously and the tasks get bigger so must find deferent solution of this problem but find suitable and reliable solutions it’s not enough as well as the evaluation of machine learning improved very fast so we compared between the ordinary and ML algorithm in the same environment, however in both case as the research related with the CPU recourses there is some sides of this problem we can work on it in the future for example the interrupt statement that could happen when the CPU working.

References

1- Aydin, M. E., & Oztemel, E. (2000). Dynamic Job-Shop Scheduling Using Reinforcement Learning Agents. Robotics and Autonomous Systems, Vol. 33, 169(178. Elsevier Science)

2- Bertsekas, D. P. (2001). Dynamic Programming and Optimal Control. 2nd edition. Athen Scientiæ

3- X. Lv, X. Gu, X. Deng, L. Zhang, and W. Li, “Resource Allocation Algorithm for VoLTE with Semi-Persistent Scheduling,” in Proc. IEEE 81st Vehicular Technology Conference (VTC Spring), pp. 1-5, 2015

4- Hui Gao and Wim H. Hesselink. A formal reduction for lock-free parallel algorithms. In Proceedings of the 16th International Conference on Computer-Aided Verification, pages 44–56, 2004.

5- Perkins, J. R., Humes, C., Kumar, P. R. (1994). Distributed Scheduling of Flexible Manufacturing Systems: Stability and Performance. IEEE Trans-actions on Robotics and Automation, Vol. 10, 133(141. IEEE Press

6- Y. Wang and I. Witten, “Induction of Model Trees for Predicting Continuous Classes,” Technical report 96/23. Hamilton, New Zealand: University of Waikato, Department of Computer Science, 1996.

7- H. Witten Ian and F. Eibe, “Data Mining: Practical Machine Learning Tools and Techniques,” Morgan Kaufmann, San Francisco, 2005.

8- K. M. Brethauer and B. Shetty. A pegging algorithm for the nonlinear resource allocation problem.
Computers & Operations Research, 29:505–527, 2002.

9- Maged M. Michael. Safe memory reclamation for dynamic lock-free objects using atomic reads and writes. In PODC ’02: Proceedings of the twenty-first annual symposium on Principles of distributed computing, New York, NY, USA, 2002. ACM.

10- Yingjie Guoa, Linzhi Wub, Wei Yuc, Bin Wud, Xiaotian Wange. The Improved Job Scheduling Algorithm of Hadoop Platform: University of Chinese Academy of Sciences, 100408, China.

11- Xiaohuang Huang1, Christopher I. Rodrigues1: XMalloc: A Scalable Lock-free Dynamic Memory Allocator for Many-coreMachines: 10th IEEE International Conference on Computer and Information Technology, pages 34-36, 2010

12- Gemma Reig, Javier Alonso, and Jordi Guitart, Prediction of Job Resource Requirements for Deadline Schedulers to Manage High-Level SLAs on the Cloud: Ninth IEEE International Symposium on Network Computing and Applications, 2010