**CPU Load Prediction & Disk Scheduling Algorithms**

Submitted By

**Aditya mehta 16BCI0184**

**Siddhant Shailaindra 16BCB0129**

**Harshwardhana Singh 16BCE0231**

**Vivek boora 16BCE0667**



Prepared for

**OPERATING SYSTEMS**

**Abstract**

The dynamic nature of a resource-sharing environment means that applications must be able to adapt their behavior in response to changes in system status. Predictions of future system performance can be used to guide such adaptations. In this paper, we present and evaluate several new one-step-ahead and low-overhead time series prediction strategies that track recent trends by giving more weight to recent data. We present results that show that a dynamic tendency prediction model with different ascending and descending behavior performs best among all strategies studied. A comparative study conducted on a set of 38 machine load traces shows that this new predictor achieves average prediction errors that are between 2% and 55% less (36% less on average) than those incurred by the predictors used within the popular Network Weather Service system.

**Keywords**

1. **Introduction**

In multi-user time-shared systems, applications are in active competition with unknown background workloads introduced by other users. The contention that results from the resource sharing tends to cause load and resource availability to vary over time. The impact can be particularly significant on a computational Grid , which links interconnected but geographically distributed computing resources as a cooperating set of resources. Clearly, predictions of system performance are necessary for efficient use of such resources. Performance predictions can be useful to both applications and schedulers. Applications can use predictions to adapt their behavior in response to changes in system status to get better performance. Schedulers can use predictions to guide their scheduling strategies and thus to achieve higher application performance and more efficient resource use. Varying CPU load has a significant effect on the running time of CPU-bound applications. Indeed, for certain types of applications the running time of a compute-bound task is linearly proportional to the average

CPU load it encountered during the execution . The focus of this paper is predicting the CPU load of shared computing resources. Our contribution is to introduce a new time series prediction technique that behaves better than other techniques used previously. Rather than giving the same consideration to the history data within a “sliding window” as do traditional linear models and mean- or median-based models, our one-step-ahead time series prediction strategies give more weight to more recent measurements than to other history data. We also allow for the use of different behaviors when “ascending” and “descending.” Our experimental results show that, on a range of host load measurement datasets, our dynamic tendency strategy with different ascending and descending behavior consistently outperforms the nine predictors used within the Network Weather Service (NWS) , a widely used performance prediction system. The rest of the paper is structured as follows. Section 2 introduces background and related work. Section 3 gives a detailed description of the prediction strategies we studied. Section 4 describes the experimental results when our prediction strategies are applied to actual measurements and compared with those of other researchers. Section 5 presents our conclusions and notes directions for further work.

1. **Literature Survey**

Previous efforts indicate that CPU load is strongly correlated over time, which implies that historybased load prediction schemes are feasible. We believe that the key to making accurate predictions is to correctly model the relationship of the history data with the future values. Time series modeling has been studied widely in many areas, including financial data prediction , earth and ocean sciences , biomedical signal processing , and networking . In the area of CPU load prediction, the Network Weather Service provides one-step-ahead predictions for any time-series fed to its predictor module. NWS is a distributed system that periodically monitors and dynamically forecasts the performance of various network and computational resources. NWS applies a collection of one-step-ahead prediction strategies to time series and chooses the strategy used for the “next” prediction dynamically according to which strategy has been most accurate over recent measurements. The prediction strategies used by NWS currently include running average, sliding window average, last measurement, adaptive window average, media filter, adaptive window media, α-trimmed mean, stochastic gradient, and autoregressive . Dynamical selection of the best prediction strategy on the fly has resulted in predictions equivalent to, or slightly better than, the best predictor in the set. We note that while for the purposes of comparison we measure the improvements achieved by our new prediction relative to those used within NWS, our new strategies could easily be included as predictors within the NWS framework. Thus, our work does not invalidate the NWS approach but rather shows that its choice of predictors can be improved. Dinda et al. evaluated multiple linear models, including autoregressive (AR), moving average (MA), autoregressive moving average (ARMA), autoregressive integrated moving average (ARIMA), and autoregressive fractionally integrated moving average (ARFIMA) models. Their results show that the simple AR model (also used in NWS) is the best model of this class because of its good predictive power and low overhead. More complex liner models are expensive to fit and hence difficult to use in a dynamic or real-time setting. Our approach, as shown in Section 4, performs better and has less overhead than these approaches.

1. **Proposed Method/System**

This section defines our prediction strategies. Each strategy predicts the one-step-ahead value based on a fixed number of immediately preceding history data measured at a constant-width time interval. We present two families of strategies:

(1) homeostatic prediction strategies, Section 3.1, and

(2) tendency-based prediction strategies, Section 3.2

* 1. **Homeostatic prediction strategies**
     1. Independent static homeostatic prediction strategy
     2. Independent dynamic homeostatic prediction strategy
     3. Relative static homeostatic prediction strategy
     4. Relative dynamic homeostatic prediction strategy
  2. **Tendency-based prediction strategies**
     1. Independent dynamic tendency prediction strategy
     2. Relative dynamic tendency prediction strategy
     3. Dynamic tendency prediction strategy (mixed variation).designed to make efficient data placement approaches on a wide area network, where good predictions determine how well an application can run.

**implementation**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithems/errors% | 5 | 10 | 15 | | 20 | 25 | 30 | 35 | 40 |
| Independent static homeostatic prediction strategy | 49% | 68% | 37.5214% | | 10.224% | 275% | 71% | 9% | 75.259% |
| Independent dynamic homeostatic prediction strategy | 24% | 43% | 22.193% | | 2.22815% | 43.5396% | 19.82% | 32.44% | 48.2 |
| Relative static homeostatic prediction strategy | 41% | 58% | 48.17% | | 7.921% | 155% | 100% | 46.3% | 62% |
| Relative dynamic homeostatic prediction strategy | 17% | 24% | 17.7457% | | 24.995% | 24% | 24% | 26% | 24.3125% |
| Tendency-based prediction strategies | / | / | / | | / | / | / | / | / |
| Independent dynamic tendency prediction strategy | 1% | 7% | | 13.56% | 11.23% | 14.22% | 24.212% | 17.321% | 55.69% |
| Relative dynamic tendency prediction strategy | 12% | 39% | | 120% | 46% | 192% | 222.13% | 100% | 122.36% |

**Results and Discussions**

We ran a series of experiments using our predictors in order to validate their effectiveness under a variety of conditions. We break these experiments into two sets. In the first set, we ran all of our predictors on a small set of time series over which we had complete control, and we evaluated the effect of different collection rates on our own predictors, on a simple last-value predictor, and on the Network Weather Service. In the second set of experiments, we ran a much larger set of 38 load traces and evaluated only our best predictor and the NWS. The last-value predictor uses the current measured value as the predicted value of the next measurement. It can be expressed by the following

formula: PT+1 = VT Harchol-Balter and Downey show that this is a useful prediction strategy for CPU resources. It has low computation and storage overhead and is the default predictor in several current systems because of its simplicity. The NWS dynamically selects the best predictor from a set that includes mean-based prediction strategies, median-based prediction strategies, and AR model-based prediction strategies. It has been shown to yield forecasts that are equivalent to, or slightly better than, the best forecaster in the set. This implies that if our prediction strategy performs better than NWS predictor, it can perform better than all the prediction techniques in the set. We did no model fitting for any of the experiments, as is commonly needed in linear regression techniques.

The parameters were defined by using training data off-line before the experiments, and were not redefined during the experiments. Thus, we minimized the run-time cost of these strategies; on average, this cost is only a few milliseconds per prediction.

**6. Conclusion**

Applications and schedulers each can benefit from accurate predictions of future resource availability when making decisions concerning how to use time-shared resources. In this paper, we have presented and evaluated two families of novel one-step-ahead time series prediction strategies that weight recent data in various ways. We presented experimental results that allow us to identify one such strategy as the best of the two families and to demonstrate that this strategy outperforms the widely used NWS predictor by 36% on average. While not every prediction is better, performance is clearly better on average. Comparison of prediction results on over 50 CPU load time series demonstrated that giving more weight to the most recent values significantly affects prediction accuracy

**7. Future Enhancements**

. Although our prediction strategy has been described and evaluated in the context of CPU load, we expect that it will also prove effective in other contexts. We plan to extend its use to network bandwidth and latency predictions. In addition, we are using this information to guide a scheduler designed to make efficient data placement approaches on a wide area network, where good predictions determine how well an application can run.

**Disk scheduling**

**Introduction**

Since the emergence of computers, hardware technology has been evolving at a tremendous pace. This evolution includes storage technologies. The current trend in storage technology is miniaturization for portability an increased storage capacity. Due to the volatile characteristic of the CPU register, Cache, and Main Memory, the use of secondary storage devices such as

Disk came into existence. In a movable-head disk, access may take the form of a

write or a read operation performed by the access arm, which holds the read/write head. Since the invention of movable head disk, the Input and Output (I/O) performance has been improved by implementing proper and intelligent scheduling of disk accesses. Disk scheduling involves a careful examination of pending requests to determine the most efficient way to service the requests. Some of Disk Scheduling algorithms are First Come First Serve (FCFS), Shortest Seek Time First (SSTF), Scan, Look, C-Scan algorithm.since look and c look can be achieved by modification in scan and c scan they have not been considered here although they are going to be taken during the result.

**Need for disc scheduling**

Disk scheduling is important because:

1. Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled.
2. Two or more request may be far from each other so can result in greater disk arm movement.
3. Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.

**Basic terminology:**

* Seek Time:

Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.

* Rotational Latency:

Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.

* Transfer Time: Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
* Disk Access Time = Seek Time + Rotational Latency +Transfer Time

**Problem statement : We have taken the following track requests for accessing the tracks as (25, 10, 151, 170, 62, 46, 74 and 111) and the initial disk head position is at 45**

1. **FCFS**

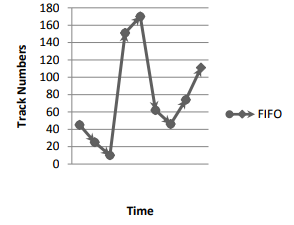
First Come First Served (FCFS) is a Non-Preemptive scheduling algorithm. FIFO (First In First Out) strategy assigns priority to process in the order in which they request the processor. The process that requests the CPU first is allocated the CPU first. This is easily implemented with a FIFO queue for managing the tasks. As the process come in, they are put at the end of the queue. As the CPU finishes each task, it removes it from the start of the queue and heads on to the next task.

Advantages:

* Every request gets a fair chance
* No indefinite postponement

Disadvantages:

* Does not try to optimize seek time
* May not provide the best possible service



Representation of FIFO

**2. STFS**

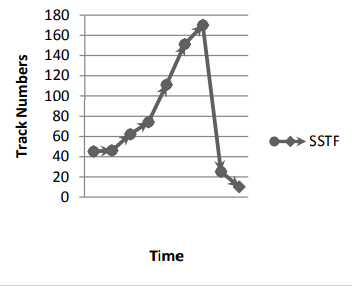
In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system

Advantages:

* Average Response Time decreases
* Throughput increases

Disadvantages:

* Overhead to calculate seek time in advance
* Can cause Starvation for a request if it has higher seek time as compared to incoming requests
* High variance of response time as SSTF favours only some requests



Representation of SSTF

**3. SCAN**

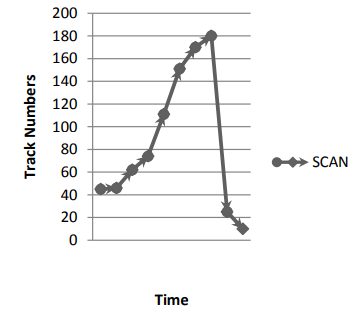
In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in it's path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works like an elevator and hence also known as elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Advantages:

* High throughput
* Low variance of response time
* Average response time

Disadvantages:

* Long waiting time for requests for locations just visited by disk arm

****

**Representation of SCAN**

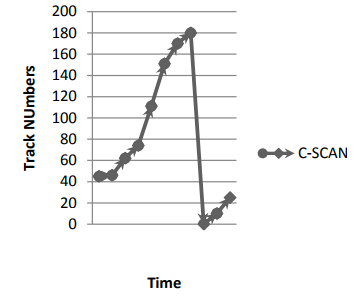
**C-SCAN**

In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.These situations are avoided in *CSAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular

fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Advantages:

* Provides more uniform wait time compared to SCAN



Representation of C-SCAN

**Comparison**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Algorithm** | **Average Seek Time** |
| 1 | FCFS | 48 |
| 2 | STFS | 29.25 |
| 3 | SCAN | 45 |
| 4 | C-SCAN | 47.5 |
|  |  |  |

**Comparision of Average seek Times of different algorithms**

**Result**

Each algorithm is unique in its own way.Overall Performance depends on number and type of requests.

**Conclusion**

 As several wild swings are experienced by the FCFS scheme, it gives the worst scheduling performance. SSTF is much better compared to LOOK (upward direction) and C-LOOK. It has also been noticed that LOOK is more efficient than C-LOOK at all loads, whereas C-LOOK is better at high loads only, as it reduces the starvation problem. The performance of each algorithm, however, depends heavily on the number and type of requests.

**C programs of the following algorithms is uploaded on this link**

**https://drive.google.com/drive/u/0/folders/0B9gXS6Mo2iy0eWNKLV9GQ3M0T00**

**8. References**

[1] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kufmann, 1998, pp. 701. Google and wikipedia

[2] M. Faerman, A. Su, R. Wolski, and F. Berman, "Adaptive Performance Prediction for Distributed Data-Intensive Applications," presented at Proceedings of the ACM/IEEE SC99

Conference on High Performance Networking and Computing, Portland, OR, 1999.

[3] S. V. Adve, A. F. Harris, C. J. Hughes, D. L. Jones, R. H. Kravets, K. Nahrstedt, D. G. Sachs, R. Sasanka, J. Srinivasan, and W. Yuan, "The Illinois GRACE Project: Global Resource Adaptation through Cooperation," presented at in the Proceedings of the Workshop on Self-Healing, Adaptive, and self-MANaged Systems (SHAMAN), 2002.