Phase Determination and Phase Analysis of Motif-based Big Data Architectures

Ishaan Lagwankar
Student, 6th Semester B. Tech in CSE
PES University
Bangalore, India
ishaanlagwankar@gmail.com

Ananth Narayan
External
ananthnarayan@gmail.com

Dr. K V Subramaniam
Centre for Cloud Computing and Big Data
PES University
Bangalore, India
subramaniamkv@pes.edu

***Abstract*—Motifs used in Big Data workloads, with a specific focus on benchmarking has allowed more abstract and generic workloads to be defined as combinations of these motifs, where each motif is defined to be predictable and deterministic in nature. We theorize that if a workload is broken into a sequence of motifs, then a measure of a performance sample can be obtained at the start of each motif. In this paper we outline novel phase detection methods, using clustering algorithms and try to identify similar workloads across benchmarking suites to quantify similar behavior, and build a neural network-based approach to detect the phase transfer of workloads in their time of execution. We identify the phase determination changes for the Sort, TeraSort, WordCount and some graph benchmarks such as MD5, PageRank and Connected Components, identified as key repeating motifs in the benchmarks under consideration.**

***Index Terms*—benchmarks, data motifs, workloads, interference-aware, phase**

I. Introduction

A motif[Gao+18b] is a combination of code, the data served as input and the pattern of the data seen by the workload monitor. It is usually assumed that each workload, or each motif rather, is predictable in nature. The advent of motifs was mainly governed by the need to build generic benchmarks, as building constrained benchmarks on non-generic workloads triggered a loss of standardisation across different suites. The idea of motifs was introduced to mitigate the non-generalised behavior of benchmarking suites, to make a novel framework to quantify each application as a combination of these motifs, and allow us to build more generic benchmarks.

Each motif inherently transfers through different phases in its execution cycle, and predicting what phase a workload transfers to from a specific point in its execution is what we aim to quantify, with the use of machine learning frameworks.The phase of the workload is built into the workload structure, and certain factors of the phase determination step could be dependent on the external factors such as an OS, the underlying hardware, and the underlying software on which the workload runs.

Our model of execution delves into understanding the key characteristics of each workload observed, by collecting data on what are the sample metrics observed, such as cache occupancy, memory usage, instruction cycle counts, and so on. This is fed into a neural network pipeline aimed to identify phase transfers, made with a recurrent neural network framework to account for series based phase transfer of workloads, with dependencies on phases already observed. The model is optimistic in a fashion that it assumes the workloads pass through only a set of observed phases, and no new phases creep in during the execution of a single motif, and also under the assumption that all these motifs run in an interference aware fashion, where a motif is not affected by any other motif or application running in the same execution environment.

In the next sections, the paper outlines first the previous work existing on the concepts of motif-based behavior, phase detection and some existing benchmarking tools we used to look at for motifs under consideration. The paper then proceeds to elucidate upon the methodology and techniques used to detect and predict phase movements. The following section provides results on the analyses performed on different workloads, with detection metrics. This is followed by our concluding remarks and justification for the results produced in a concise fashion.

II. Previous Work

Wanling Gao et. al [Gao+18a] talks about the complexity and diversity of big data and AI workloads, and how understanding them is difficult and challenging. The paper proposes the idea of Data Motifs, to allow a lens into understanding how big data applications consist of discrete motifs executing in sequences defined by the type of workload under observation. It outlines the use of BigDataBench (publicly available from http://prof.ict.ac.cn/BigDataBench), in identifying and plotting execution motifs, implemented in the framework. The framework identifies eight data motifs, that include Matrix, Sampling, Logic, Transform, Set, Graph, Sort and Statistic, and provides a comprehensive analysis and characterization of these data motifs from the perspective of data sizes, data flow, sources, patterns and helps understand how these motifs model benchmarking suites currently in use.

We examine the build of many benchmarking suites, but we selected some commonly used benchmarking suites for the basis of this experiment

A. BigDataBench

BigDataBench[Gao+18a] is a realistic, modern benchmarking suite which has the capability to run not only collectively as a whole end-to-end application differentiate time breakdown of modular approaches in programs, but also in a Microbenchmarking perspective, for fine tuning kernels and hot spot functions. It provides 13 representative real-world datasets and 27 big data benchmarks, consisting of 6 workload types including online services, offline analytics, data warehousing, NoSQL, and discrete stream processing workloads. It also identifies microbenchmarks of single motifs, component benchmarks, which are combinations of data motifs, and end-to-end application benchmarks, which are combinations of component benchmarks. It also provides the BigDataGeneratorSuite, use to generate synthetic data to run these benchmarks.

B. HiBench

HiBench[Hua+10] is a benchmarking suite that allows the MapReduce model of large scale data analytics to be used as a benchmarking framework, for benchmarking applications running on Hadoop. It provides synthetic microbenchmarks and real-world Hadoop applications, used to characterize the Hadoop framework in terms of speed, throughput, HDFS bandwidth, system resource usage and data access patterns. It identifies the microbenchmarks of Sort, WordCount and TeraSort for Hadoop execution, which were relevant to understanding the differences in the build of HiBench and BigDataBench, and allowed us to compare and contrast the nuances that made the execution paradigm differ in the two benchmarking suites.

C. CRONO

CRONO[Ahm+15] is an open source graph benchmarking suite for shared memory multicore machines. It consist of graph workloads, with both synthetic and real world graph inputs. It provides the benchmarks of Path planning, with contains the Single Source Shortest Path, All Pairs Shortest Path and Betweeness Centrality workloads. It also provides the Search workloads of Breadth first search and Depth first search, accompanies with the Travelling Salesman benchmark. It also encompasses Graph processing workloads of Connected Components, Triangle counting, PageRank and Community Detection.

We use predominantly the HiBench and BigDataBench benchmarking suites for our purposes, with their common benchmarks as metrics for our applications.

III. Methodology

The approach outlined can be divided into 3 discrete parts, where the first part entails the data collection and metric observations, the second defines the clustering method used to outline phases of the motif execution, while the third defines the prediction algorithm used to predict phase transitions.

A. Data Collection

The data collection phase involved understanding what metrics to observe and calculate, and to analyse their usage and importance in clustering workloads into phases of executions. The different metrics observed were mainly collected through the perf tool and the sar tool for CPU monitoring and profiling workloads. The different architectural events measured were

- Instruction Count
- Branch Instruction Count
- Branch Mispredictions
- Instructions per Cycle
- LLC Reference Count
- LLC Miss Rate
- Cycle Count
- DTLB Miss Count
- ITLB Miss Count
- LLC Occupancy
- Total Bytes Used
- Local Bytes Used

These metrics were standardised for 1000 (kilo) instructions to enforce readability and inferencing. We used the perf tool for profiling these, for all the workloads in BigDataBench and HiBench.
We proceeded to identify upon what dataset sizes to profile these workloads, and came to an acceptable metric for each benchmarking suite to identify what constituted a small, medium and large sized dataset to profile these metrics, provided in Table 1.

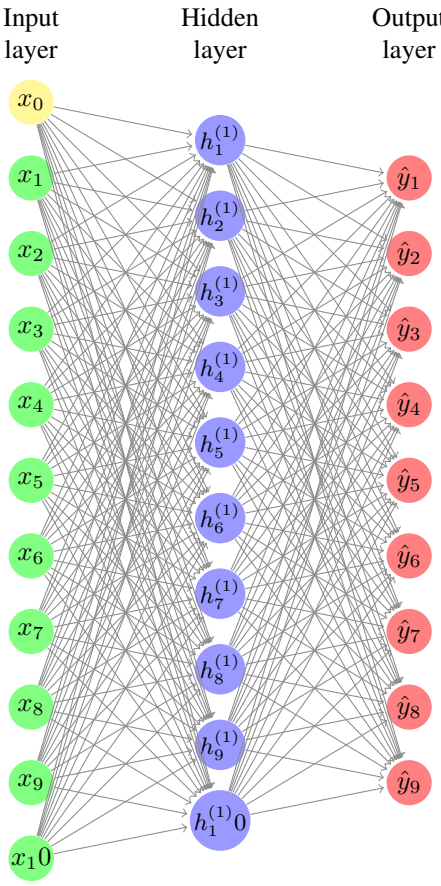
B. Clustering and Phase Classification

Once the sizes were identified, these workloads were run and profiles were collected in an interference-aware environment, where no workload was affected by the runs of any other system application, taken as a baseline.

A Naive clustering algorithm was run to identify clusters for each datapoint.Clustering was done using K means clustering, considering 9 clusters[Fig: 1]. We applied PCA to reduce dimensionality to 2 components for the sake of visualisation, but the whole number of features was used for the clustering metric.

C. Phase Transition Predictions

Once the results for the phase classification based on phase behavior clustering were obtained, we appended the cluster transition phases at each timestamp to create a clear picture of when a data point changes cluster. We then used a feed-forward neural network to determine when each data point is capable to transfer to another phase. The network we used, however, did not account for a time-series sensitive model, but was not necessary as the network was able to predict the correct phase transitions for the data obtained. The network is described below.



IV. Results

The hardware used to simulate the collection of this data was as follows:

1) 32 core Intel Machine
2) 64 GB RAM
3) KVM used for VM isolation
    a) 32 GB RAM
    b) 4 vCPUs

We used data over 3 hadoop versions, to simulate differences in underlying software frameworks and see differing workload metrics. For this we used

1) Hadoop 2.10.0
2) Hadoop 3.2.1

First, the data points for clustering are shown here for an example Sort workload, to illustrate the different clusters formed when these workloads were run. The data points were normalised by adding per-kilo (1000) instructions for each metric and the data was pooled into one large dataset.

What we observed from this naive clustering is that every workload spends the same amount of time in a certain phase and transfers phase at the same time, if the time intervals are normalised over the total execution time, regardless of dataset size, as observed in Sort[Fig: 2], TeraSort[Fig: 3] and WordCount[Fig: 4].

Although this result is propagated consistently in the Hadoop 3.2.1 workloads, similarity between different versions of Hadoop cannot be drawn with confidence, as seen in Fig: 5. This result is yet to be quantified, and represents the future scope of the project, to analyse how versional changes affect the execution cycles of these workloads.

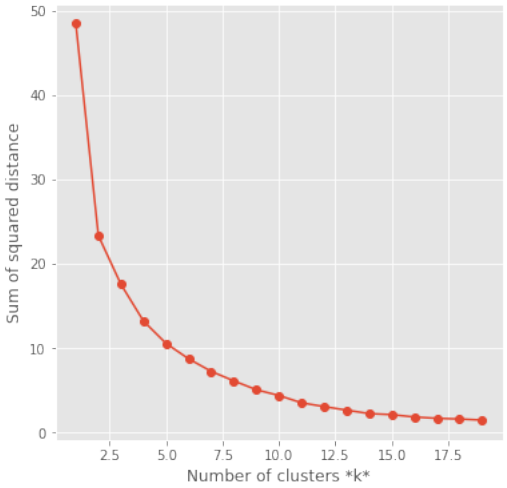


Fig. 1: SSE over clusters

The feed-forward neural network was built in mind to consider all the factors of the data point, and not just the current cluster in which the data point resided. We obtained an accuracy of 92% in the prediction cycle, and gave accurate results for our validation data, allowing us to determine the behavior of each data point, based on cluster metrics.

V. Concluding Remarks

We were able to achieve phase determination and prediction in a very naive fashion, giving us an accurate picture as to how a workload changes phase behavior and how to quantify this behavior in terms of time spent in the total execution cycle. However, we do acknowledge the existence of drawbacks, being mainly the simplicity of the model used to quantify these metrics. Had we used some sort of time-based metric to quantify these relationships, such as a Hidden Markov model or a Recurrent Neural Network, it may have boosted the predictions to be better, but we kept the simplicity due to the sole reason that such complex models need not be used to data that inherently needs very simple classification metrics, as the clustering and predictions are not complex to draw. Out future work involves exploring these different techniques to see whether they boost our prediction accuracy and work with more complex graph workloads to quantify their behavior, with suites such as LDBC[Ang+20], and other benchmarking suites such as MLPerf[Red+19].

Acknowledgment

This project would not be complete without the continued efforts of all the team members and our guide, Dr. K V Subramaniam, with the unparalleled dedication of the Centre for Cloud Computing and Big Data for continued efforts poured into refining the project. We would also like to thank PES University, and its Computer Science and Engineering

TABLE I: Data Size Specification

Workload	Small Spec	Medium Spec	Big Spec
Sort	32000 bytes	3200000 bytes	320000000 bytes
TeraSort	32000 bytes	3200000 bytes	320000000 bytes
WordCount	32000 bytes	3200000 bytes	320000000 bytes
APSP	Threads: 32 Vertices : 12000 Degrees : 2	Threads: 32 Vertices : 30000 Degrees : 800	Threads: 32 Vertices : 60000 Degrees : 1000
BC	Threads: 32 Vertices : 1000 Degrees : 2	Threads: 32 Vertices : 3000 Degrees : 500	Threads: 32 Vertices : 6000 Degrees : 5900
BFS	Threads: 32 Vertices : 1000 Degrees : 2	Threads: 32 Vertices : 30000 Degrees : 10000	Threads: 32 Vertices : 90000 Degrees : 50000
Community	Threads: 32 Vertices : 1000 Iterations: 40 Degrees : 200	Threads: 32 Vertices : 10000 Iterations: 50 Degrees : 4000	Threads: 32 Vertices : 80000 Iterations: 50 Degrees : 2000
Connected Components	Threads: 32 Vertices : 1000 Degrees : 200	Threads: 32 Vertices : 50000 Degrees : 900	Threads: 32 Vertices : 100000 Degrees : 1000
DFS	Threads: 32 Vertices : 10000	Threads: 32 Vertices : 500000	Threads: 32 Vertices : 1000000
Pagerank	Threads: 32 Vertices : 10000 Degrees : 5000	Threads: 32 Vertices : 50000 Degrees : 12000	Threads: 32 Vertices : 100000 Degrees : 40000
SSSP	Threads: 32 Vertices : 1000 Degrees : 400	Threads: 32 Vertices : 50000 Degrees : 8000	Threads: 32 Vertices : 100000 Degrees : 9000
Triangle Counting	Threads: 32 Vertices : 1000 Degrees : 800	Threads: 32 Vertices : 5000 Degrees : 2000	Threads: 32 Vertices : 100000 Degrees : 9000
TSP	Vertices : 10	Vertices : 40	Vertices : 50

Department for the opportunity to work on this project and provide exposure to new-age concepts through its efforts.

REFERENCES

[Hua+10] Shengsheng Huang et al. "The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis". In: Apr. 2010, pp. 41–51. DOI: 10.1109/ICDEW.2010.5452747.

[Ahm+15] Masab Ahmad et al. "CRONO: A Benchmark Suite for Multithreaded Graph Algorithms Executing on Futuristic Multicores". In: *2015 IEEE International Symposium on Workload Characterization* (2015), pp. 44–55.

[Gao+18a] Wanling Gao et al. *BigDataBench: A Scalable and Unified Big Data and AI Benchmark Suite*. 2018. arXiv: 1802.08254 [cs.DC].

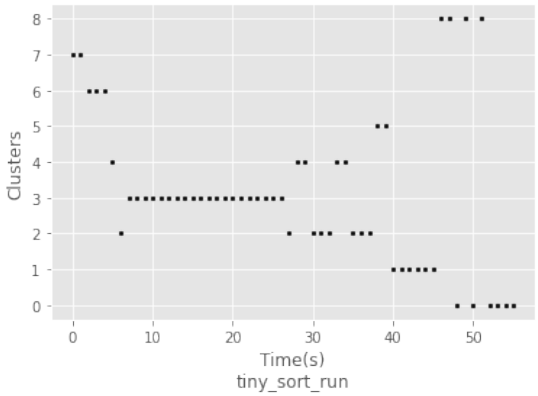
[Gao+18b] Wanling Gao et al. *Data Motifs: A Lens Towards Fully Understanding Big Data and AI Workloads*. 2018. arXiv: 1808.08512 [cs.DC].

[Red+19] Vijay Janapa Reddi et al. *MLPerf Inference Benchmark*. 2019. arXiv: 1911.02549 [cs.LG].

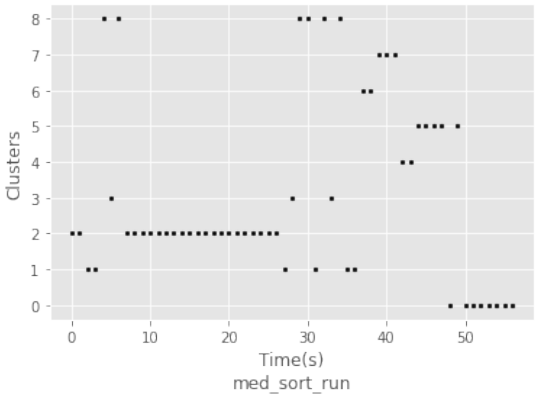
[Ang+20] Renzo Angles et al. *The LDBC Social Network Benchmark*. 2020. arXiv: 2001.02299 [cs.DB].

APPENDIX

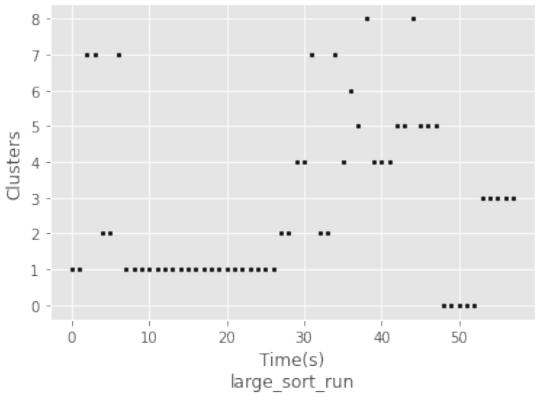
The graphs for the Sort and WordCount workloads are displayed as follows, as reference to the observations, along with comparison graph example for Sort across Hadoop versions.



(a) Small

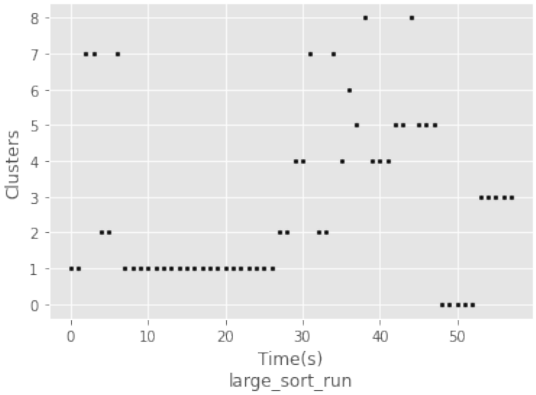


(b) Medium

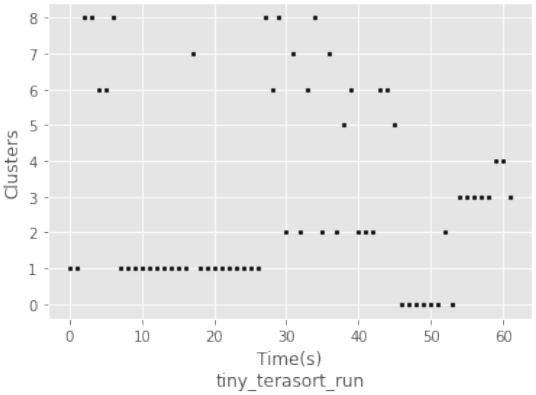


(c) Large

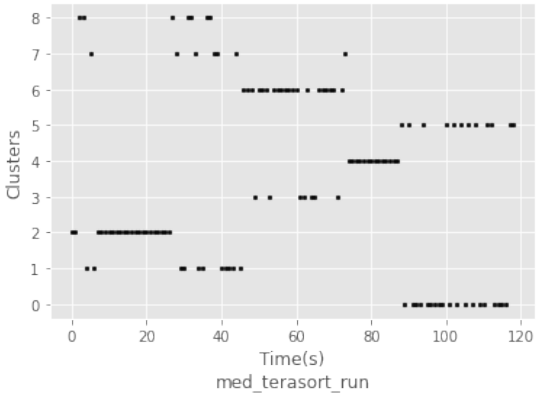
Fig. 2: Sort Clustering



(a) Small

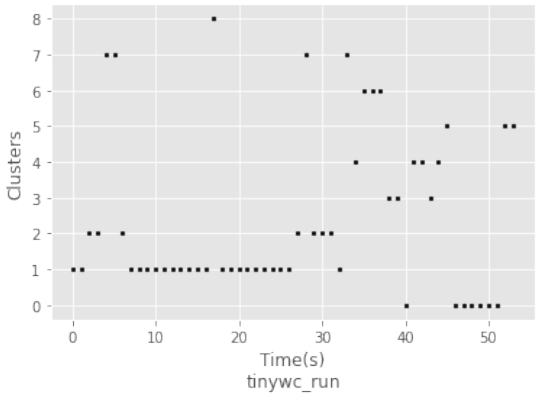


(b) Medium

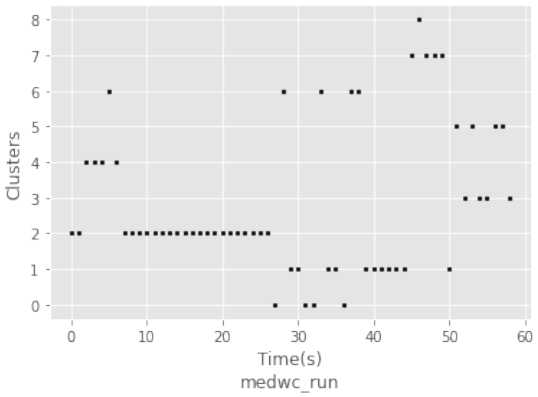


(c) Large

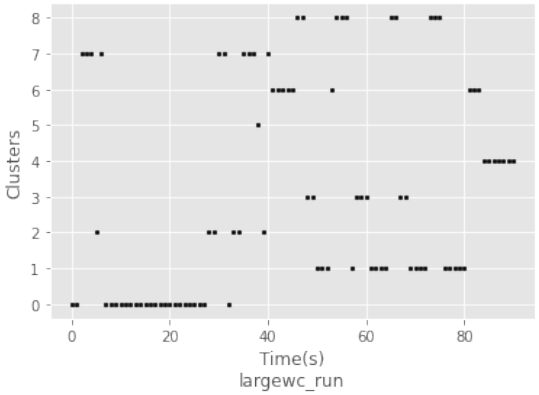
Fig. 3: TeraSort Clustering



(a) Small

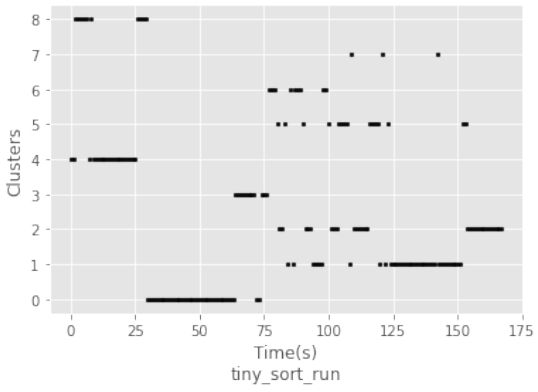


(b) Medium

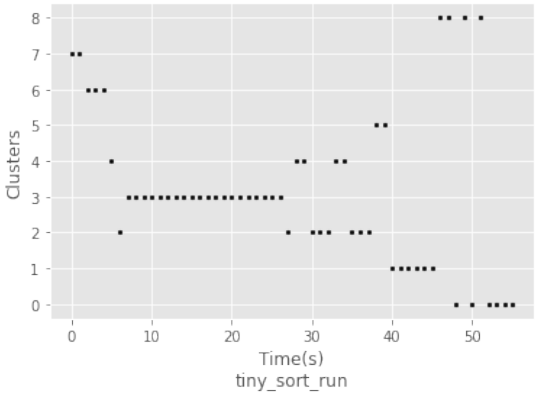


(c) Large

Fig. 4: WordCount Clustering



(a) Hadoop 2.10.0



(b) Hadoop 3.2.1

Fig. 5: Comparison of Sort Across Hadoop versions