A Workload Aware Model of Computational Resource Selection for Big Data Applications

Amit Gupta, Weijia Xu
Texas Advanced Computing Center
University of Texas at Austin
{agupta,xwj}@tacc.utexas.edu

Natalia Ruiz-Juri, Kenneth Perrine
Center for Transportation Research
University of Texas at Austin
nruizjuri@mail.utexas.edu, kperrine@utexas.edu

***Abstract*—Workload characterization of Big Data applications has always been a challenging research problem. Big data applications often have high demands on multiple computing components in concert, such as storage, memory, network and processors and have evolving performance characteristics along with the scale of the workload. To further complicate the problem, the increasing diversity of hardware technologies available makes side-by-side comparisons hard. Choosing right resources among a wide array of available systems is a decision that is likely to plague both end users and resources providers. In this paper, we propose a workload aware model for the computational infrastructure selection problem for a given application. Our model considers both features of the workload and features of the computational infrastructure and predicts expected performance for a given workload, based on historical performance results using Support Vector Machines (SVM). We tested our model with a practical application from the domain of Transportation research on two distinct computing resources. The application has significant requirements on both memory availability and processing power. Therefore the optimal performance of the application is a dedicated trade-off between different types of resources and it is workload specific. The two testing systems represent two main trends in high performance computing resources. One infrastructure is a traditional high end computing cluster consisting of moderate number of CPUs and memories running at high frequency and high bandwidth. The other system, based on the latest Intel Knights Landing processor, is a good representation of the trending Many-Core technology in which high number of processing cores running at lower frequencies are available. The memory allocation models are also often different between the two systems. Our results show that our proposed model can achieve over 90% accuracy in performance prediction with small training data sets for our test application. The results also indicate that our model is a viable approach to be extended to other classes of applications and to be potentially adopted by high performance computing resource providers.**

***Keywords*-Workload Characterization, Big Data Application, High Performance Computing, Many-core, Network Modeling, Intel Knights Landing, Support Vector Machine,**

I. Introduction

The era of Big Data has catalyzed explorations of various sciences and natural phenomena at large scale and brought a wide class of applications. With the advent of Internet of Things [37] and ubiquitous sensors, this trend of new workloads accompanying more data is only going to rapidly increase. Despite efficient algorithmic implementations, Big Data applications still require significant amount of computational resources to handle ever increasing amount of data and continues to stimulate new advances in hardware technologies. However, as both computing infrastructure is getting more diverse and big data workloads are getting more complicated, choosing the right computational resource and efficiently utilizing the state-of-the-art hardware for big data applications becomes an increasingly challenging question. Here, we present a novel workload aware model that can be utilized for computational resource selection for a given big data application.

Workload characterization of the big data applications is an emerging problem, driven by various goals (e.g energy conservation, optimal resource utilization etc) and has brought several investigation and discussions over the years [7], [36], [18], [31]. There are two primary challenges in estimating the performance of a big data application. The first is the high resource requirements of big data application on several computational components instead of just on processors or storage units. For example, the parallelism of a big data application is not simply bound by number of cores but also on the memory resources, data transfer rate between the different computing components. Therefore, the collective subsystems of a computing infrastructure must be considered jointly rather than independently. The second challenge is the fact that the workload characteristics and performance bottlenecks are themselves moving targets and can evolve with the scale of the data being analyzed. For example, the effect of data access that may not significant for smaller data sets but becomes greater for larger scale can be out weighted by the computing requirement if the data keeps growing. Therefore, it makes old methods of performance estimation based on smaller scale of data becoming less accurate and reliable.

Furthermore, the availability of diverse new computing hardware technologies developed for big data applications presents a new issue for the end users to select the most appropriate computing resources. Due to the intense quest towards maximal performance both in academia and industry, hardware manufacturers have strategically responded with ad-hoc specialized hardware tailor made for application classes. We see this in the form of general purpose graphic processing units (GPGPUs) leveraged towards highly parallel mathematical operations and specialized flash storage technologies leveraged for quicker data ingest. Furthermore, with Moores Law [20] causing a noticeable upper bound in the achievable speed of a single core, we're seeing the beginnings of availability of many-core designs in hardware and faster memory pipelines [14], [5], [35]. Even if the workload requirements of the application is known, mapping those requirements to hardware features and utilizing them efficiently still remains as a challenging task.

Our work is directly motivated by the question: how should we choose amongst equally applicable computational resources available to run a big data application? Our goal is different from performance benchmarking of a software application or highly granular profiling of a specific workload for analysis. The proposed model is not to make direct prediction for the performance but to infer which infrastructure is better suited for a given analysis task. The model can directly benefit both resource providers by improving resource utilization and data analysts by better facilitating their analysis jobs.

Our model utilizes support vector machine (SVM) [12] to build a prediction model based on the historical performance of an application on the various computational resources being considered. The prediction model considers both hardware specifications of the computation resources and variations in the data sets to be analyzed. In this paper, we detail our proposed model, preliminary implementation and evaluations using an application in the Transportation research field with real world problem requirements. For this test case, our model can achieve over 90% accuracy in predicting the right system to use. The core computation in this application is a distributed implementation of finding shortest paths over a set of pairs of nodes for large network. Therefore, the results presented here may also be generalized to a larger class of similar applications in network graph analysis. The results also indicate our model could be a viable approach to be applied with other applications and can be extended to a very large infrastructure deployments and application profiling scenarios.

II. Background and Related Work

In this section, we describe research related to work presented here and backgrounds about our testing application and systems. In computer science, our work is related to the research on both performance benchmarking [16], big data systems [18], [38] and workload characterization [7], [36], [31].

A. Performance Benchmarking and Workload Characterization

With various software stacks and technology for high performance computing prevalent for use in big data, there is a lot of interest benchmarking big data systems performance. For instance, many efforts like SparkBench [18] and HiBench [13] revolve around performance benchmarking for frameworks based on the Hadoop ecosystem for generic computing (Spark [38]) and data warehousing (Hive [34]). There are also efforts on performance improvement of applications on specific hardware such as for GPGPU and Xeon Phi [28], [22], [8]. However, the common approaches in performance benchmarking are based on the foundation of well established benchmark workloads. The central goal of those benchmarks is to provide a complete and yet unbiased measure of the system performance. However, the significance of these benchmark test numbers to the performance improvements of actual deployed applications, both in online commercial services and academic scenarios, is still somewhat obscure.

Another set of relevant work is on the topic of workload characterization [32], [31], [7]. These works largely attempt gain a very granular picture of the effects of their application workload on the underlying hardware by means of monitoring application activity with low level tracing tools and hardware performance counters. Our work differs from the workload characterization in that we are not focused on a particular data set or optimizing the internal workings of a specific analysis task. Our model is to identify a generic approach to estimate relative performance among different computing infrastructure for a specific user analysis task.

B. The Workload : Dynamic Traffic Assignment

In our preliminary study presented here, we used an application for Dynamic Traffic Assignment (DTA) in transportation research as our test case. The application is chosen for several reasons. First, it is a good representation of a big data application with all characteristics described in Section (1). The application has both data intensive and computing intensive elements. Therefore, the practical analysis requirements of this application varies greatly which in turn creates variations in its performance profile. Secondly, the core computation steps are relatively simple and common to many other network analysis. Lastly, the authors have extensive knowledge and experience with this distributed computation code development and running various workloads.

DTA models are powerful tools for realistically representing complex transportation networks of urban regions involving millions of daily interactions amongst travelers and between travelers and the transportation infrastructure. Simulation based DTA models e.g., [21], appropriately account for the impact of traffic control strategies, the propagation of congestion, and the presence of tolls and turning movement delays, and have seen increasing adoption by transportation network planners and operators in the last decade. A crucial component in DTA models is the computation of time-dependent shortest-paths (TDSP) [26], [10] and is often the most computationally demanding portion of the model. The efficiency of transportation systems and infrastructure, investigated by models like DTA, make them very relevant to modern life. Additionally, Transportation networks and their shortest path problems are a (complex) variant of graph based iterative processing workloads. Numerous Big Data problems are formulated as iterative computations run on graphs (ex PageRank [27], Query processing in graph databases [1], Network Community Detection [24]). The necessity of DTA workflows to scale across computational infrastructure to enable potentially impactfull investigations is also one shared by several Big Data problems from other domains. DTA models therefore belong to a domain of contemporary investigative

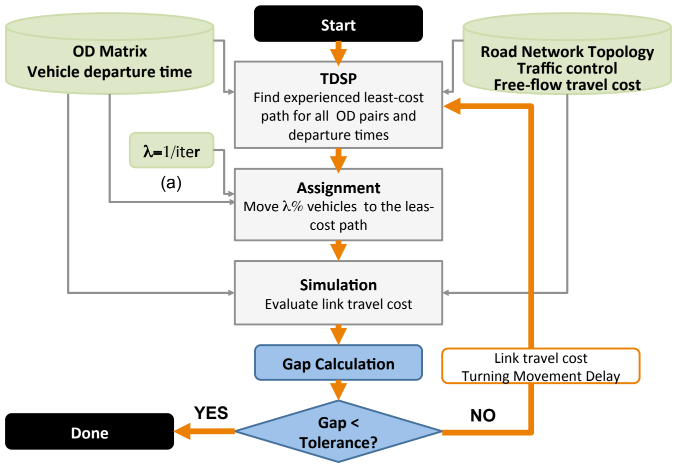

Fig. 1: Basic execution flow of a DTA algorithm.

research and bears critical relevance to modern society, while also sharing some ”structural” similarities to problems from other domains makes them a very good exemplar of Big Data applications. We have chosen to use a DTA implementation named VISTA [25] as it is a mature, stable and extensible DTA implementation that has been widely used in Transportation research [17], [3].

DTA methodologies typically [29] run stages iteratively as show in 1. Time Dependent Shortest Path (TDSP) Computations are a complex variant of standard shortest path algorithms in graphs, that compile a list of shortest paths for all origin-destination pairs of interest. They are solved by extension of standard $O(n \cdot m)$ label correcting algorithm to account for time varying link costs that reflect traffic movement and infrastructure changes. Traffic dynamics themselves are driven by a table of ”demand” or desired origin-destination trips at specific times. Demand tables are furnished by a combination of synthetic and actual data gathered by city Department of Transportation organizations on real traffic patterns. Assignment is a stage when the appropriate vehicles are assigned shortest paths from those found in TDSP. The Simulation stage is largely single threaded and is a discrete simulation of each vehicle travelling through the network, making choices based on minimizing cost and responding to congestion and infrastructure conditions. These simulation dynamics in-turn change the cost of links (i.e congestion) in the roadway network. At the end of this process, the program measures to see if traffic conditions have reached an ”equilibrium” state that is measured by a threshold (for details refer [4]). For large, city-sized geographic regions, these processes are very demanding on computational infrastructure and can be time consuming to execute.

C. Advances in Computing Infrastructure

Our preliminary evaluations were conducted using two different latest computing infrastructure. One infrastructure is the traditional multi-core architecture commonly used in the computing cluster and high end workstation (Stampede [33]). And the other is a latest many-core architecture which features an order of magnitude more processing cores (based on Intels latest Knights Landing processors).

These two types architecture quite accurately represent the dichotomy of hardware choices that are likely to plague Big Data researchers in the future. The multi-core architecture offers the user, a few cores at a faster clock speed. In contrast however, many-core architecture offers the user much more cores (by an order of 17x more) albeit at a slower clock speed. Additionally many-core architecture offers the user, access to faster (but limited to 16G) memory. A workload that can be re-factored to be massively parallel will almost certainly run faster on many core systems. However, its serial portion will run faster on the quicker multi-core chips. The differences in memory management, availability, access mechanism and system bandwidth also directly affect the performance of big data application.

There are several different implementations and hardware products that utilize Many-core architecture including GPGPU, Xeon Phi etc.[19], [6]. Here we focus on the uses of the latest Intels Many Integrated Core (MIC) architecture, the Intel Knights Landing processors (Intel Xeon Phi 7250, refered as KNL thereafter). Each KNL node has a total of 68 cores with 4 hardware threads per core to a total of 272 (1.4GHz) hardware threads. Each node also has 96G of memory, 16GB of which is a faster Multi Channel Dynamic Random Access Memory (MCDRAM) technology. The supporting network interconnect uses Omnipaths 100Gb/sec network. While traditionally, many-core designs were commonly used for the narrow purpose of offloading computations, as a co-processor (e.g Intel Knights Corner[6], GPGPUs [11]) , the KNL serves as the primary host CPU of the compute node. This marks a sharp change in the trend of CPU architectures available in servers today and is likely to become more prevalent from other hardware manufacturers in the future as well. With the 2 types of memory available, the KNL also has different modes of operation where the memory hierarchy can be modified at boot time while being transparent to the end user (eg. memory mode where the faster MCDRAM is used as a large L3 cache, cluster mode where different internal mechanisms for cache coherency may be selected etc).

In comparison, the multi-core system used here is based on multi-core Intel Xeon E5 Sandy Bridge processors. Each compute node has 2 Intel Xeon E5-2680 Sandy Bridge Processors. These amount to a total of 16 (2.7GHz) hardware threads. Furthermore each node has 32G of memory. The supporting network interconnect uses Mellanox FDR InfiniBand technology in a 2-level (cores and leafs) fat-tree topology.

We should also expect to see other types of specialized systems equipped with faster storage subsystems [15] and data pipelines that may also be adequate for consideration by the Big Data engineer in the future. For brevity however, we illustrate using just 2 types of hardware highlighted above while noting that our analysis can be extended in real deployment situations to multiple infrastructure types.

III. A Workload Aware Model for Resource Selection

A. Supply and Demand Model of Computing Resource

The basic idea of our approach is to predict the relative performance for different computing resources for a given analysis task based on historical performance data of the same application. Our model follows the basic supply and demand model [9]. Here the available computational resource is the supply while the computation resources required by the application is the demand. When the supply can meet the demand, we assume the application can be executed at its maximum efficiency. Otherwise, the application cannot be run at its optimal settings. Furthermore, when the supply is more than demand, which means additional computing resources will not reduce the execution time, it indicates a low utilization of the computational resources.

$$f(dem, sup) = \left\{ \begin{array}{cc} \text{low utilization} & \text{sup} > \text{dem} \\ \text{optimal} & \text{sup = dem} \\ \text{low efficiency} & \text{sup} < \text{dem} \end{array} \right\}$$

Applying to a specific computing subsystem, we can assume the time required (as a measure of the performance) is a continuous function of the supply and demand before the supply can meet the demand and is a constant when the supply meets the demand.

$$t_s(dem, sup) = \left\{ \begin{array}{cc} \text{t}_{min} & \text{sup} \geq \text{dem} \\ \text{g(dem, sup)} & \text{sup} < \text{dem} \end{array} \right\}$$

We should point out that we are disregarding cases of internal code optimization of application workloads, where given the same hardware resources, performance of an application may be improved by innovations in employing better algorithms, internal data structures and improving access patterns between computation and data in general. All of these fall in the purview of the application designer/developer and outside the scope of a generic tool. For the purposes of this paper, we assume that such exercises in performance optimization have already been explored.

The total time of an analysis task required using a given computing resource can be modeled as the sum of the time spend on all the subsystems.

$$T_{total} = \sum t_s(dem, sup)$$

For a specific analysis, the performance (i.e. the execution time) can be treated as a function of the available computing resources, (including number of computing cores, core frequencies, available memory, data transfer rates between different subsystems) and the analysis requirement (including memory usage, storage usage, and computing operations). While the hardware specification can be easily obtained and remain constant for a given computing resource, the exactly demand measure varies and is specific to each analysis tasks. Therefore, we further assume that the demand is a function of the input parameters of the specific analysis. The input parameters should also include direct measures of the input data. Therefore, the equation 3 can be

$$T_{total} = \sum t_s(d(input\_para), s(hw\_specs))$$

Although there are many subsystem within the computing node, we only consider two components here, the computing power and memory. This is mainly due to the fact that two computing architecture are available within the same computing cluster and share the same configurations for other subsystems. Therefore the processing cores and memory are two most differences between the two in our cases. The testing application uses a number of parameters, we choose the set of measures that can best to reflect the workload. A major variation between different analysis request is indeed the network topology and number of origin and destination nodes considered. Therefore, we've chosen following features to capture both the characteristics of the supply and the demand in our use cases.

- Hardware - Processing : (Total Number of Threads used by the job across all nodes) $\times$ (Speed of one core)
- Hardware - Memory : Total memory available to the job across all nodes.
- Problem Size - Graph Topology : Nodes in the road network
- Problem Size - Graph Topology : Links in the road network
- Problem Size - Computation size : Number of unique Origin-Destination pairs in the demand table
- Problem Size - Simulation Dynamics : Number of Vehicles in the simulation
- Problem Sizes - Simulation Dynamics : Number of trips in the simulation

B. Prediction Based on Historical Data

Instead of trying to derive the exact model relation and parameter for this particular application, we would like to derive a general approach to estimate relative performance between different resources and workloads. We choose using SVM for model training and predication. Support Vector Machines (SVM) [12] is a class of supervised learning models that use algorithms to classify input data in one of two output classes/groups. Their goal divide the input points in their spacial dimension by a hyperplane that separates the two groups by the widest margin possible. Internally SVM's may employ many different kernels or methods to measure distance between two points in their spatial dimension. The ones most commonly used, which are Linear, Polynomial, Radial Basis and Sigmoid (or hyperbolic tangent). These essentially serve as different similarity measures and replace the dot product operation in the SVM algorithm. They are primarily useful in extending the classification using SVM's into non-linear spatial dimensions [2]. The nature of performance characteristics of workloads, their input and the underlying hardware events are seldom linear due the complexity of all the layers

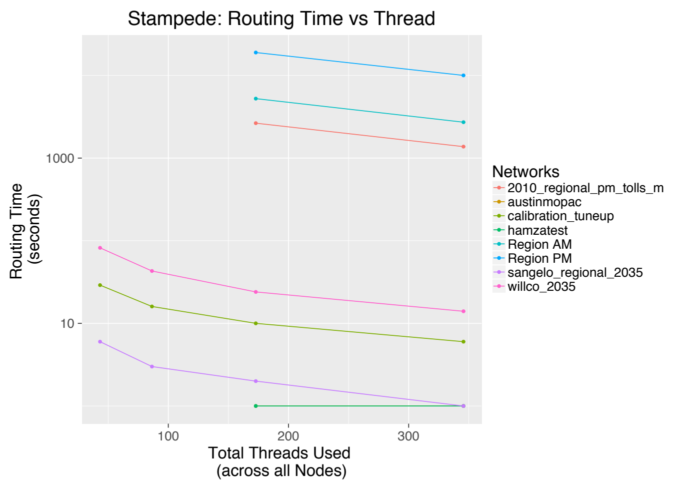

Fig. 2: Stampede Scaling with Number of Cores

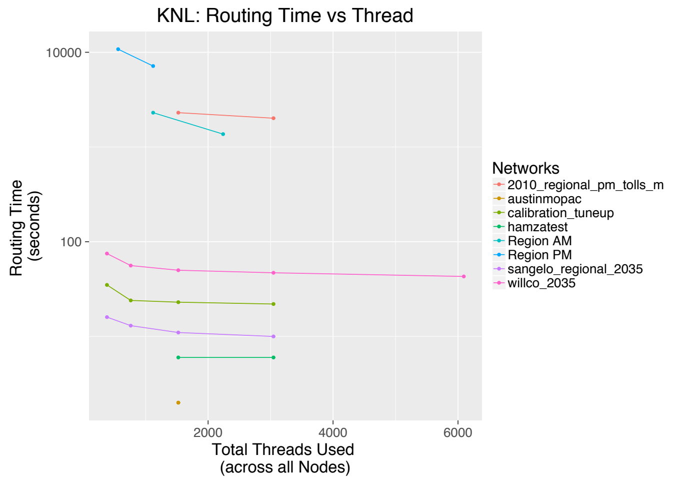

Fig. 3: KNL Scaling with Number of Cores

of software/hardware. We therefore empirically explore these SVM variants in our classification model.

To train our SVM model, we recorded the running time between the two systems under various hardware configurations. We then created two classes based on which system and configuration has better performance for the same workload.

IV. Evaluation Results

A. Data and Testing Environment

To train our model on the 7 features we selected from Problem Size and Hardware, we ran DTA models on various Transportation networks described in the table below. We attempted to cover a wide range of network topology, vehicles/trips simulated and number of Origin-Destination pairs. Please refer to Table (I) for a list of these networks.

We ran these networks on two types of nodes, multi-core nodes and KNL nodes, on Stampede cluster. Each KNL node has a total of 68 cores with 4 hardware threads per core to a total of 272 threads. Each node also has 96G of memory, 16GB of which is a faster MCDRAM technology. The supporting

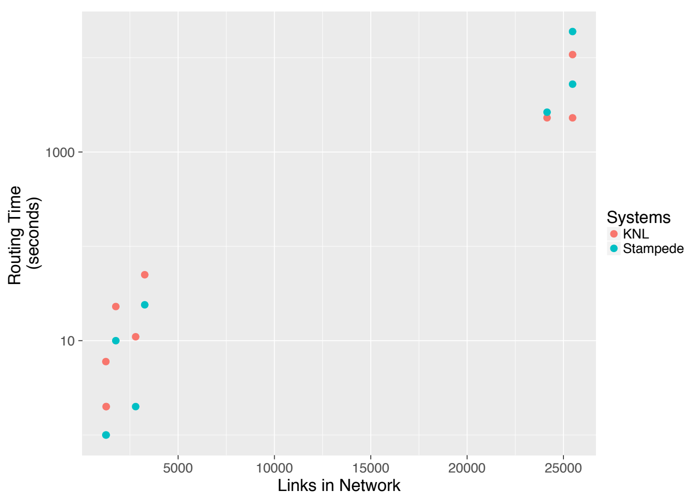

Fig. 4: 4 Node: Runtime scaling with Links in graph

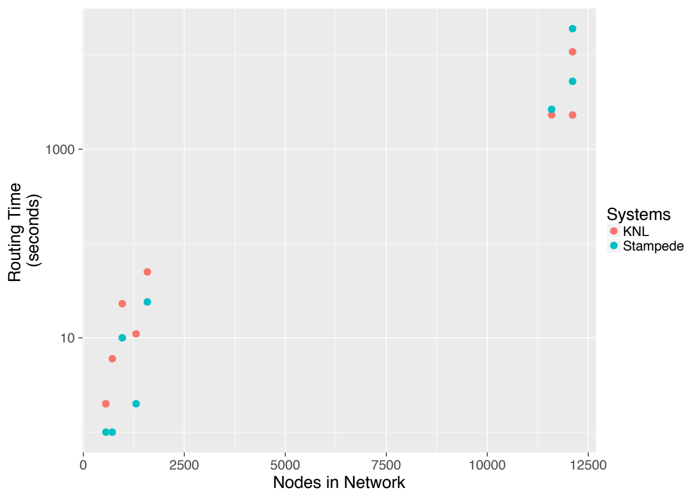

Fig. 5: 4 Node: Runtime scaling with Nodes in graph

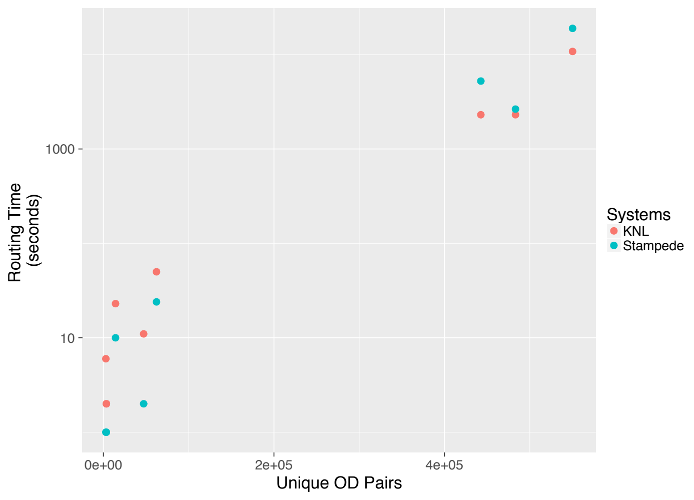

Fig. 6: 4 Node: Runtime scaling with Unique OD Pairs

Network	Nodes	Links	O-D pair	Vehicles	Trips
calibration	964	1767	14244	27251	256256
sangelo	1306	2797	47216	136090	72737
willco_2035	1585	3268	62290	196067	587166
region_am	12112	25473	442606	7326169	810437
region_pm	12112	25473	550147	7352025	1275782
hamzatest	546	1251	2902	99	63316
austinmopac	558	1267	3518	1	88855
2010_regional_pm_tolls_m	11594	24149	483213	7211650	1096619

TABLE I: Transportation Networks data

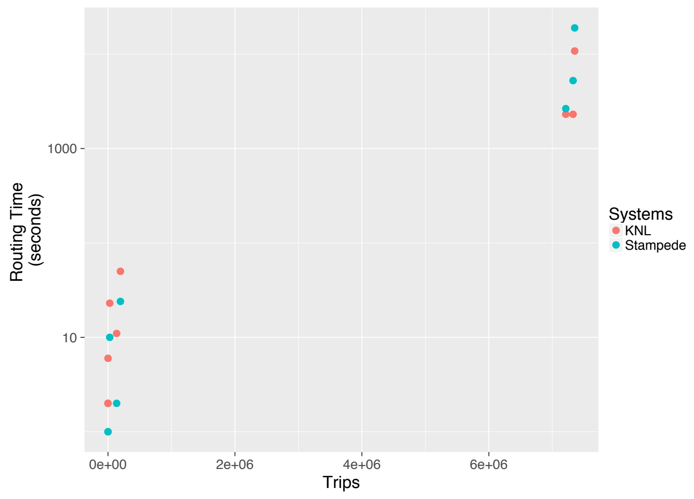

Fig. 7: 4 Node: Runtime scaling with Trips

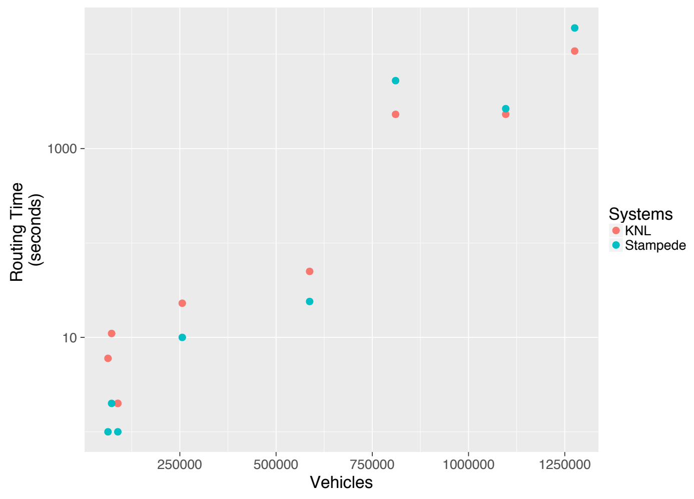

Fig. 8: 4 Node: Runtime scaling with Vehicles

network interconnect uses Omnipaths 100Gb/sec network. The multi-core system sued here is based on multi-core Intel Xeon E5 Sandy Bridge processors. Each compute node has 2 Intel Xeon E5-2680 Sandy Bridge Processors. These amount to a total of 8 (2.7GHz) cores (16 threads). Furthermore each node has 32G of memory. The supporting network interconnect uses Mellanox FDR InfiniBand technology in a 2-level (cores and leafs) fat-tree topology.

B. Performance Profiles

We did node for node comparisons between Stampede and KNL systems for scaling with both processing power (in Figures (2), (3)) and various dimensions of the input problem size, namely graph nodes and links (Figures (4), (5)), Unique Origin-Destination pairs, (Figure (6)), Trips (Figure (7)) and Vehicles (Figure (8)). These figures shown are for testing with 4 compute nodes. A trend that becomes quickly evident from these figures is that neither system is universally better for all problem sizes. We see that in the case of larger networks, KNL does almost upto 2x better and takes 50% lesser time. These networks, in addition to being larger topologically, also deal with greater traffic volume injected into them, thus creating more dynamics in the variation of link costs. This in-turn causes TDSP to become more computationally intensive, which is helped by running on many-core hardware for larger number of threads. Smaller networks on the other hand benefit less from such large number of threads and instead benefit from faster cores of traditional multi-core systems. Testing results for the 8 compute node case were also similar. This further motivates the need for our model as given a problem instance, it is often difficult to judge which system it might perform better on.

C. prediction model evaluation

To validate our approach, all testing results are pooled together to form the training and evaluation benchmark. In this benchmark, each sample record include all features on hardware specification and testing workload listed in previous section. For each sample record, we can assign a binary classification as ”preferred” or not ”preferred” based on the run times of iteration 2 in DTA algorithm. Since we have a small testing data sets with 21 samples, we conducted multi-iteration cross validation methods for evaluating model efficiency. In each iteration, the benchmark is randomly split into two subsets as training and evaluation data sets. During the sample split, we also guarantee that both classes are selected proportionately from either subset. Two splits ratios are used: one with 80/20 (17 samples/4 samples) splits and the other with 70/30 splits (16 samples/5 samples) between training and evaluation subset. For each split, we run 10,000 trails.

Our preliminary testing used the SVM support implemented in e1071 package in R [30][23]. There are four kernel functions supported in this implementation including: Linear,

SVM Kernel	70% (Training)-30% (Testing)	80% (Training)-20% (Testing)
Linear	93.568	93.01
Polynomial	95.198	92.7425
Radial Basis	95.494	96.1725
Sigmoid	94.906	95.3925

TABLE II: Prediction accuracy of SVM Kernels

Polynomial, Radial Basis and Sigmoid. We conducted the same trail using four kernel separately. In summary, there are 80,000 total cross validation tests conducted between the four kernel functions and two different splits. The average accuracy results, defined as the percentage of true positive in the testing sets, are reported in Table (II).

The results shown that our approach can achieve over 92% accuracy for all test scenarios. The accuracy is slightly higher when using 80/20 split than using 70/30 split. For both splits, the best kernel function is the Radial Basis function. As we can see from these results, our methodology works well. At this point we would like to acknowledge that our data set size is small. However, the results shows our method is promising and could be used for resource selection for DTA analysis with high confidence level. The results are also very encouraging on the general feasibility of our approach for additional types of applications and hardware specifications.

V. Future Work & Conclusion

In this paper, we presented a new model to help users selecting computational resources for specific analysis tasks. Our model utilizes the historical data collected from the same application but with different workloads running across the available resources. We used a SVM classifier and DTA analysis in transportation research as the workload during our preliminary studies. We showed that the test application performance profiles can change over the workload and availability of the resources non-linearly. Our results shows high prediction accuracy and confirms its viability for this model. The best performing SVM kernel was found to be the Radial Basis function.

For this particular use case, our results also show the advantages and disadvantages between two main stream computing architectures available today: multi-core and many core. For this specific use case, our results show there is no universal advantage between these two architectures. The advantage is infact workload dependent. We firmly believe the performance profile for most big data applications will also share this characteristic. Therefore, research and tools of workload aware models of application and systems are necessary.

Our work is directly motivated by the gap between the increasing diversity of types of high performance computing resources becomes available and the increasing complexity in performance profile of big data applications. Both types of hardware are available at the same computing clusters at Texas Advanced Computing Centers. Choosing between the right type of nodes is a practical question for both the resources providers and the cluster users. Our work not only has the potential to directly benefit the cluster users to accomplish their computation task sooner but can also benefit resources providers to improve resource utilization.

We attempted to use all available real transportation data (instead of synthetic data) to adequately capture the feature values that would form a realistic representation of the problem space. However, the present available labelled data are very small. In the future, we aim to be able to both gather more real world transportation simulation data and also to explore the qualitative differences in using synthetic data sets for model training. We also plan to expand our testing to additional applications from other fields as well. Although hundreds of applications were run repeatedly in our cluster, a challenge is to collect metrics for input parameters used by users due to both technical and policy issues.

In the future, we plan on exploring this methodology within an application specific web portal which allows users to launch different instantiations of the same application. This would enable us to constantly gather more data to train the SVM model on and thereby produce increasingly accurate predictions to the portal users. We also hope to incorporate more hardware features (e.g L2/L3 caches, network interconnect technology, memory bandwidth) to capture the effect of these lower level hardware components into the model to improve its prediction accuracy.

Acknowledgment

All computations are run using Stampede cluster at Texas Advanced Computing center funded by National Science Foundation. The work is also funded in part by XSEDE project. The authors would like to Thank the Center for Transportation Research for access to the simulation framework and network data.

References

[1] Pablo Barceló Baeza. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 175–188. ACM, 2013.

[2] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[3] Ozge Cavusoglu, Virginia P Sisiopiku, and Natalia Ruiz Juri. Role of transit in carless evacuation planning. *Natural Hazards Review*, 14(3):191–199, 2012.

[4] Yi-Chang Chiu, Jon Bottom, Michael Mahut, Alex Paz, Ramachandran Balakrishna, Travis Waller, and Jim Hicks. Dynamic traffic assignment: A primer. *Transportation Research E-Circular*, (E-C153), 2011.

[5] George Chrysos. Intel® xeon phi coprocessor-the architecture. *Intel Whitepaper*, 2014.

[6] George Chrysos and Senior Principal Engineer. Intel xeon phi coprocessor (codename knights corner). In *Proceedings of the 24th Hot Chips Symposium*, 2012.

[7] Kapil Dev, Xin Zhan, and Sherief Reda. Power-aware characterization and mapping of workloads on cpu-gpu processors. In *Workload Characterization (IISWC), 2016 IEEE International Symposium on*, pages 1–2. IEEE, 2016.

[8] Yaakoub El-Khamra, Niall Gaffney, David Walling, Eric Wernert, Weijia Xu, and Hui Zhang. Performance evaluation of r with intel xeon phi coprocessor. In *Big Data, 2013 IEEE International Conference on*, pages 23–30. IEEE, 2013.

[9] David Gale. The law of supply and demand. *Mathematica scandinavica*, pages 155–169, 1955.

[10] Amit Gupta, Weijia Xu, Kenneth Perrine, Dennis Bell, and Natalia Ruiz-Juri. On scaling time dependent shortest path computations for dynamic traffic assignment. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 796–801. IEEE, 2014.

[11] Tianyi David Han and Tarek S Abdelrahman. hicuda: High-level gpgpu programming. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):78–90, 2011.

[12] Marti A. Hearst, Susan T Dumais, Edgar Osman, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[13] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *New Frontiers in Information and Software as Services*, pages 209–228. Springer, 2011.

[14] James Jeffers and James Reinders. *Intel Xeon Phi coprocessor high-performance programming*. Newnes, 2013.

[15] Christopher Jordan, David Walling, Weijia Xu, Stephen A Mock, Niall Gaffney, and Dan Stanzione. Wrangler's user environment: A software framework for management of data-intensive computing system. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 2479–2486. IEEE, 2015.

[16] Jithin Jose, Sreeram Potluri, Karen Tomko, and Dhabaleswar K Panda. Designing scalable graph500 benchmark with hybrid mpi+ openshmem programming models. In *International Supercomputing Conference*, pages 109–124. Springer, 2013.

[17] Michael W Levin, Matt Pool, Travis Owens, Natalia Ruiz Juri, and S Travis Waller. Improving the convergence of simulation-based dynamic traffic assignment methodologies. *Networks and Spatial Economics*, 15(3):655–676, 2015.

[18] Min Li, Jian Tan, Yandong Wang, Li Zhang, and Valentina Salapura. Sparkbench: a comprehensive benchmarking suite for in memory data analytic platform spark. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, page 53. ACM, 2015.

[19] Dmitry I Lyakh. An efficient tensor transpose algorithm for multicore cpu, intel xeon phi, and nvidia tesla gpu. *Computer Physics Communications*, 189:84–91, 2015.

[20] Chris A Mack. Fifty years of moore's law. *IEEE Transactions on semiconductor manufacturing*, 24(2):202–207, 2011.

[21] Hani S Mahmassani and KF Abdelghany. Dynasmart-ip: Dynamic traffic assignment meso-simulator for intermodal networks. *Advanced Modeling for Transit Operations and Service Planning*, pages 201–229, 2003.

[22] Julio Daniel Carvalho Maia, Gabriel Aires Urquiza Carvalho, Carlos Peixoto Mangueira Jr, Sidney Ramos Santana, Lucidio Anjos Formiga Cabral, and Gerd B Rocha. Gpu linear algebra libraries and gpgpu programming for accelerating mopac semiempirical quantum chemistry calculations. *Journal of chemical theory and computation*, 8(9):3072–3081, 2012.

[23] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Friedrich Leisch Andreas Weingessel and, Chih-Chung Chang, and Chih-Chen Lin. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group*, 2015. version 1.67.

[24] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 815–824. ACM, 2015.

[25] NMC. Vista documentation, June 2014. Available at https://nmc-compute1.ctr.utexas.edu/vista/doc.

[26] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3):607–625, 1990.

[27] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[28] Sreepathi Pai, Matthew J Thazhuthaveetil, and Ramaswamy Govindarajan. Improving gpgpu concurrency with elastic kernels. *ACM SIGPLAN Notices*, 48(4):407–418, 2013.

[29] Srinivas Peeta and Athanasios K Ziliaskopoulos. Foundations of dynamic traffic assignment: The past, the present and the future. *Networks and Spatial Economics*, 1(3-4):233–265, 2001.

[30] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[31] Zujie Ren, Xianghua Xu, Jian Wan, Weisong Shi, and Min Zhou. Workload characterization on a production hadoop cluster: A case study on taobao. In *Workload Characterization (IISWC), 2012 IEEE International Symposium on*, pages 3–13. IEEE, 2012.

[32] Yakun Sophia Shao and David Brooks. Isa-independent workload characterization and its implications for specialized architectures. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 245–255. IEEE, 2013.

[33] TACC. Stampede documentation, October 2016. Available at https://portal.tacc.utexas.edu/user-guides/stampede.

[34] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[35] Pedro Valero-Lara, Poornima Nookala, Fernando L Pelayo, Johan Jansson, Serapheim Dimitropoulos, and Ioan Raicu. Many-task computing on many-core architectures. *Scalable Computing: Practice and Experience*, 17(1):32–46, 2016.

[36] Avani Wildani and Ian F Adams. A case for rigorous workload classification. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, pages 146–149. IEEE, 2015.

[37] Feng Xia, Laurence T Yang, Lizhe Wang, and Alexey Vinel. Internet of things. *International Journal of Communication Systems*, 25(9):1101, 2012.

[38] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.