Sprint Ability: How Well Does Your Software Exploit Bursts in Processing Capacity?

Nathaniel Morris, Siva Meenakshi Renganathan and Christopher Stewart, *The Ohio State University*

Robert Birke and Lydia Chen, *IBM Research Zurich*

***Abstract*—Computer systems constrain their processing rates to stay within power, cost and answer quality budgets. Sprinting mechanisms increase processing rates by exceeding budgets for short bursts before reverting back to safe processing rates. Sprinting mechanisms can speed up query processing and reduce queuing delay, but it is challenging to set policies on when and how hard to sprint. This paper discusses *sprint ability*, i.e., performance under a set sprinting policy divided by the best performance achieved by any competing policy. System managers can use sprint ability to diagnose slowdown caused by poor sprinting policies. As sprinting mechanisms proliferate, system managers will need tools to measure and manage sprint ability, creating new research problems. For example, new techniques to model response time under various sprinting mechanisms and policies will be needed. Approaches to adapt sprinting policies as workloads and underlying systems change will also be needed. For this paper, we provided a first step toward these problems by building a simulator that models response time for Internet services, as an efficient mean to explore the large parameter space of sprinting policies. We set up our simulator to capture key features of sprinting policies, i.e., sprinting frequency and magnitude, studied in recent papers: ApproxHadoop and Adrenaline. In our tests, the policies proposed in those papers achieved only 71% and 75% sprint ability. Further, the difference between best and worst policies varied across sprinting mechanisms. These results confirm the need for research on techniques to efficiently manage sprinting mechanisms.**

I. Introduction

Internet services can process queries faster by over clocking processors, exceeding circuit breaker capacity and returning lower quality answers. These solutions can have harmful long-term side effects, but they can be used safely in short bursts. *Sprinting mechanisms* use such techniques to boost processing rates for short periods and then safely disable them before their side effects arise. Recent examples of sprinting mechanisms in research and commercial products include:

- Adrenaline [9] uses DVFS to boost processor clock rates, exceeding the system power budget.

- ApproxHadoop [7] drops map tasks to speed up map-reduce computations, lowering the quality of final answers.

- Data center sprinting [19] temporarily over subscribes data center circuit breakers.

- Redundant query scheduling [1], [4], [12] starts redundant query executions, increasing cost per query.

- Intel TurboBoost [15] increases per-core clock rates above their rated operating frequency if the whole processor is below power limits.

Sprinting mechanisms can speed up query executions that demand heavy computation. Also, sprinting mechanisms can subdue workload surges that would otherwise cause queuing delays. However, sprinting mechanisms can improve response time only if they can be triggered. Resource management software must preserve precious bursts in processing capacity for moments where they are needed most.

Recent research proposes various policies to manage sprinting mechanisms, termed simply sprinting policies hereon. Across the board, the proposed sprinting policies perform better than policies that do not use sprinting mechanisms. Many of the proposed policies also improve upon naive, greedy policies, showing the importance of good resource management policies. However, it is also important to explore a large space of possible management policies to answer the following research questions:

- How much better are the best sprinting policies compared to proposed sprinting policies?

- Do good sprinting policies share conspicuous features that make them easy to identify?

- Are the best sprinting policies affected by factors that change, e.g., failures, query arrival rates or renewable energy. If so, what autonomics are needed to adapt sprinting policies over time?

This paper discusses *sprint ability*, a metric that assesses sprinting policies, i.e. how well a sprinting mechanism is managed. Sprint ability divides performance achieved under a targeted sprinting policy by the best performance achieved under any sprinting policy. If sprint ability equals 1, the targeted sprinting policy manages its sprinting mechanism better than (or equal to) all other sprinting policies. Sprint ability complements performance debugging tools. It separates the speedup provided by sprinting mechanisms from the limitations of sprinting policies. To be sure, a sprinting mechanism that supports larger bursts and power budget will improve performance but not necessarily sprint ability. To improve sprint ability, resource management software must (1) use bursts more opportunely or (2) better balance the magnitude and duration of processing bursts.

The baseline for sprint ability is the best performance achieved under any sprinting policy. We argue for new simulation and modeling tools that output expected speedup given workload and sprinting factors. These tools will predict performance in a fraction of the time required to set up the system, configure the target sprinting policy and run tests.

However, these tools present new research challenges. First, a core assumption in queuing theory (a widely used approach to model response time) is contrary to sprinting: Service time is supposed to be independent of queuing delay. In contrast, many sprinting policies invoke sprinting mechanisms only when queuing delay is large. For example, Adrenaline triggers DVFS when queries are within 50% of SLO limits [9]. Correlations between service time and queuing delay break queuing theory models, making closed-form results hard to obtain. To obtain response time across a wide range of sprinting policies, we designed a sprinting-aware simulator for simple M/M/k[1] Internet services. Our simulator considers workload factors (e.g., arrival and service rates) and sprinting factors (e.g., speedup from sprinting and sprinting frequency). It uses discrete-event simulation to model queuing and total response time per query, under a given sprinting policy and mechanism. The simulator is an efficient mean to explore large parameter space of sprinting policies and search for the optimal performance that is used to compute the sprint ability.

We used our simulator to evaluate the sprint ability of internet services under two sprinting mechanisms. The first mechanism speeds up query executions by 1500X but only 25% of query executions can trigger the mechanism in a 5-minute interval. This mechanism was inspired by Approx-Hadoop [7]. Query executions are akin to map tasks that are dropped to speed up execution. For a given query, the sprinting policy in ApproxHadoop drops a fixed number of maps (i.e., 25%) as set by the user. We explored alternative sprinting policies where the drop rate changes at the end of each 5-minute interval. Our results showed that the sprinting policy that drops the same number of maps in each interval has sprint ability of 71%, and sprinting policies that drop 49% of maps once a day maximize performance.

The second sprinting mechanism speeds up query execution by up to 1.57X, but it has a limited energy budget. Sprinting policies choose how often to trigger the mechanism. This mechanism was inspired by Adrenaline [9]. We studied the proposed policy that triggers sprinting when (1) the query execution time neared SLO limits or (2) the query execution time fell above the 85^{th} percentile. In our tests, this policy achieved 75% sprint ability. With more degrees of freedom, these tests highlighted the wide ranging effects of sprinting policies.

This paper generalizes sprinting mechanisms as ephemeral processing bursts triggered by software. The hardware and software techniques used to burst processing capacity are orthogonal to the sprinting policies that determine when and how to sprint. It is our position that *there are rich research problems in setting and adapting sprinting policies.*

The remainder of this paper is as follows. Section II compares three simple policies. The discussion reveals challenges in setting sprinting policies. Section III defines sprint ability formally and makes the case that sprint ability is hard to measure without new research on performance models and system profiling. Section IV presents a simulator that provides a first step toward new models on the effects of sprinting policies. Section IV-A uses the simulator to consider sprinting mechanisms similar to ApproxHadoop and Adrenaline. Our results suggest that there is room to improve recently proposed policies. Section V discusses related work.

[1]M/M/k is a system with k servers in which the arrival and service rates are exponentially distributed.

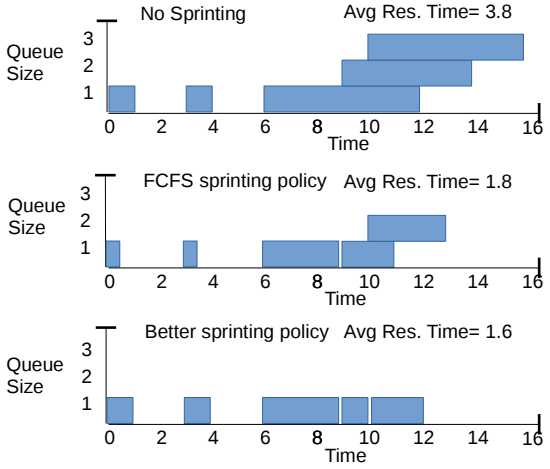

Fig. 1: Depiction of a sprinting mechanism that speeds up indivudal query executions. We compare response time under policies that (top) never trigger the sprinting mechanism, (center) greedily trigger via first come first serve and (bottom) trigger more opportunely.

II. Motivating Example

In this paper, a sprinting mechanism is hardware or software that allocates additional resources for processing queries. Dynamic voltage frequency scaling (DVFS) under an energy budget is a concrete example. It allocates additional voltage for short periods. Sprinting policies control how often and when to trigger these mechanisms. The terminology borrows from earlier operating systems concepts [13].

Figure 1 depicts query execution times, queue lengths and response time for an Internet service with access to a DVFS sprinting mechanism. Here and throughout the paper, we characterize sprinting mechanisms as a triplet: target workload, speedup per sprint and budget. In Figure 1, the DVFS mechanism targets individual query executions, speeds up their processing rate by 2X and has a budget of 4 sprinting seconds.

Figure 1 compares 3 policies that govern sprinting for the same query arrival workload. The top chart depicts query executions under a policy that never triggers DVFS sprinting. Queuing delay increases average response time to 3.8 seconds per query. The center chart depicts a policy that triggers sprinting for queries as they arrive until budget is exhausted. This fist-come-first-serve approach wastes resources on queries that execute quickly without sprinting. Finally, the bottom chart shows a policy that opportunely sprints on the third and fourth arriving queries. By triggering DVFS at the right time, this approach reduces query processing and eliminates queuing delay. Response time is 2.3X and 1.12X faster.

Figure 1 shows two key challenges in setting sprinting policies: (1) detecting which moments provide the greatest

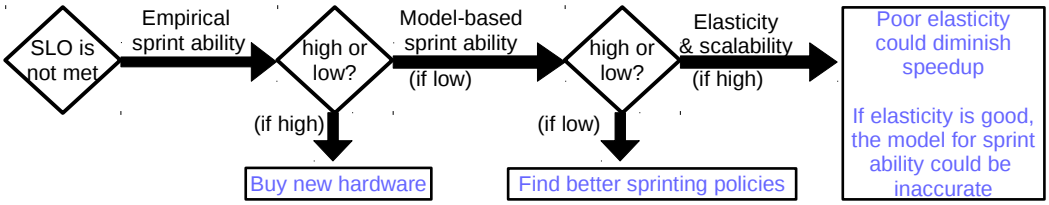

Fig. 2: Using sprint ability, empirical and model-based, to debug performance problems.

utility for sprinting and (2) carefully preserving budgeted resources for only those moments. Poor detection arises from poor scheduling or uncertainty regarding future demands. Resources are wasted when sprinting is unavailable due to triggering too often or inefficiently using resources.

III. Sprint Ability: Vision and Research Challenges

Resource management software can use only one sprinting policy at a time. The policy used should significantly reduce response time, increase throughput or, more generally, perform well on key performance metrics. Empirical approaches measure performance directly to rank competing policies. Prior research relied on such empirical methods to highlight poor detection and wasteful triggering in simple policies [9], [12], [19]. Empirical approaches are time consuming when there are many competing policies, for example, resource management software can define a new policy by adjusting tunable parameters. As a result, assessing the optimality of sprinting policies becomes even more challenging. We advocate that sprint ability offers an efficient metric to evaluate and improve sprinting policies across a wide range of sprinting mechanisms.

Particularly, sprint ability narrows competing policies by answering the following questions for a targeted policy: (1) Does the targeted policy exploit bursts in processing capacity well? (2) Are there better policies available? Equation 1 formally defines sprint ability as performance under a target policy (P_{trg}) divided by the best performance under any policy (P_{opt}). $\vec{c}$ is a collection of sprinting mechanisms available to competing policies. Each is defined by workload, speedup and budget.

$$SA = \frac{Perf(P_{trg}, \vec{c})}{Perf(P_{opt}, \vec{c})} \quad (1)$$

In our vision, P_{opt} is obtained from models or simulation. (If P_{opt} can be obtained from direct measurements, then empirical ranking remove the need for sprint ability.) Models and simulators can compute expected performance across a wide range of sprinting policies much faster than empirical approaches. There are two ways to compute P_{trg}. Empirical sprint ability (ESA) uses direct measurements. Model-based sprint ability (MSA) use models or simulations. ESA and MSA are both useful to debug performance problems with sprinting policies. *Sprint ability does not replace response time, throughput or other first-class metrics.* We plan to use sprint ability to complement performance debugging in the presence of a sprinting mechanism. Specifically, sprint ability determines whether poor performance is caused by inopportune triggering or insufficient sprinting speedup.

Figure 2 illustrates performance debugging with sprint ability. Assume an SLO violation has occurred. If ESA is high (i.e., greater than 95%) then replacing the sprinting policy will provide only small performance gains. More powerful sprinting mechanisms are needed. If ESA and MSA are both low, the sprinting policy is causing unneeded slow down. Some modelling approaches may suggest competing policies to explore for ESA. If ESA and MSA disagree, other metrics can uncover the difference between model expectations and actual performance. For example, systems with poor elasticity may over state speedup by discounting adaptation time. A wide range of tools exist to align model expectations to actual performance [2], [3], [17]. The remainder of the section outlines research in managing and measuring sprint ability.

Systems and Adaptive Resource Management: It is important to show that sprint ability integrates nicely with existing systems. We would like to characterize exactly which pieces of code in today's widely used platforms yield high sprint ability. However, modern systems are complex, comprising multiple layers (e.g., platform, OS and networking stacks). Each layer may comprise thousands of lines of code. The relevance of systems code to sprint ability depends on the sprinting mechanism. For example, code about data sampling has more relevance to software-driven approximation bound by quality limits than to dark silicon sprinting limited by power.

Table 1 provides examples of systems issues that affect sprint ability under different categories of sprinting mechanisms. While the taxonomy is not complete, it outlines interesting challenges for the autonomic computing field. For example, in software-driven approximation, fixed data sampling across all queries is similar to FCFS policy described in Section II. Policies that use variable, time changing rates could use approximation opportunely. In Section IV-A, we provide early results that confirm this opportunity. This problem along with tail detection are concerned with improving choices about when to sprint. Research on these topics directly improve sprint ability.

Sprint ability also improves when sprinting mechanisms are preserved for longer periods. In Table 1, we highlight slow core detection from Apache YARN. The problem is that core scaling spends a portion of its energy budget when systems software can't use it. Both policy and mechanism can attack this problem. Policies that target workloads in early stages, before the resource manager provisions nodes, can improve detection speeds—increasing sprint ability. In contrast, YARN could better support core scaling as a sprinting mechanism. This research will improve speedup from sprinting

Mechanism	Systems layer	Task	Challenge	Description
software-driven approximate computing	runtime platform	data sampling	detect when sprinting will have impact	sampling at a fixed rate for all queries does not minimize queue delay
	OS & distributed file system	key-value lookup	preserve resources	sampling data hosted on remote servers may limit speedup
	coordination service	resource containers	preserve resources	changing a query's sampling rate on the fly demands mechanisms to track execution over distributed resources
dark silicon	OS & runtime platform	thread manager	preserve resources	OS and runtime platforms are slow to detect new cores, limiting speedup
	runtime platform	detect slow queries	detect when sprinting will have impact	heavy query arrival rates should encourage frequent sprinting
cloud burst	distributed stores	spot market provisioning	detect when sprinting will have impact	when spot prices decrease, sprinting frequency can safely increase

Tab. 1: Open research problems for high sprint ability in adaptive resource management.

but could improve or degrade sprint ability. Looking closely at Equation 1, we note that such research extends the vector of candidate constraints, i.e., a larger collection of sprinting mechanisms in $\vec{c}$. If this increases P_{opt} by more than P_{trg}, sprint ability will decrease—even though speedup improves. Research that improves the speedup of sprinting mechanisms is best evaluated using performance improvement (not sprint ability). However, it is important to measure sprint ability whenever there are major speedups as the decision making process may need to change fundamentally.

Models for Optimal Performance: Sprint ability needs to model the best performance across a wide range of policies. Building accurate models is challenging. However, our charactization of sprinting mechanisms (workload, speedup, budget) reduces complexity. For this paper, we discuss two possible research solutions. The first and ideal solution would be closed-form queuing models that capture the effects of sprinting. These models would consider the speedup gained from sprinting and the decision making process for sprinting. However, effective sprinting that targets reduced queuing delay breaks the following fundamental assumption underlying most queuing theory models: "the presence of a long queue is supposed to have no effect on the speed of service. [11]" Queue-dependent service times complicate Markovian analysis. Multi-level queues could yield closed form for certain sprinting conditions, e.g., coarse grained server setup times [5]. Further, replication for predictability and redundancy also offer some sprinting benefits, albeit with limited choice in query targeting. We call for continued research on such models. The second solution is to build accurate simulators. To this end, Section IV presents a first-cut and open-source simulator for sprinting targeted at query executions.

Profiling Speedup Due to Sprinting: Speedup due to sprinting is hard to measure. As profiling software must know when sprinting is on. This requires advanced context tracking. Some sprinting mechanisms, such as approximation, the performance with sprinting off simply requires extending query execution with sprinting on [10]. However, other mechanisms require two totally separate query executions. Another approach uses statistical models to understand average speedup.

IV. First Steps: A Simulator

We built an M/M/k like simulator that supports a wide range of sprinting mechanisms and policies. In addition to arrival and departure rate parameters, the simulator accepts speedup and budget parameters that describe a sprinting mechanism. In its current version, our simulator only supports mechanisms that accelerate specific query executions. The simulator accepts parameters on the frequency with which sprinting policy triggers sprinting mechanisms, including support to target specific types of queries (e.g., tail response time).

Figure 4 illustrates our proposed discrete-event simulator. Prior to the first discrete time step, the simulator creates an array of query objects. A query is defined by its arrival and service time. Arrival time is relative to the first tick (0 time). Service time is absolute. In M/M/k mode, the simulator increments the clock by one, checks for an arrival, then checks the queue for a departure. In sprinting mode, the simulator executes sprinting policies at each time step. It marks whether the query execution triggered a sprinting mechanism and when its execution ended. After the simulation, it checks to see if the policy would have violated budgeted power, quality loss or cost by accelerating too many queries for too long.

Limitations: As a first step, our simulator allows us to explore the effects of a wide range of policies. However, simulation is much slower than analytic models. Closed-form models for response time with sprinting should eventually replace this simulator. Our team has already begun working on such models. However, queuing theory models are fundamentally harder when the service rate depends on the queue size. Ghandhi et al. needed multi-level hidden Markov models to build queuing models for elastic services where the queue size adjusts the number of servers slowly over time [5]. However, sprinting is harder to model because the increase in processing

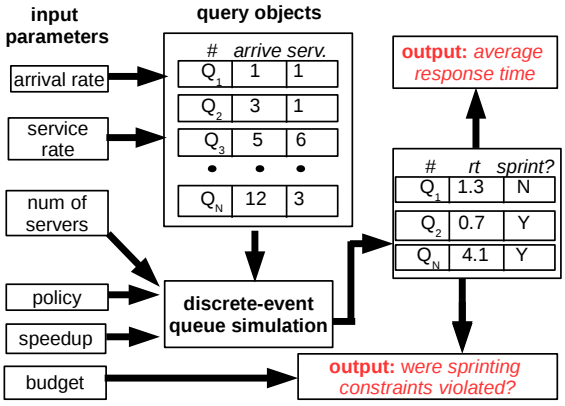

Fig. 4: Architecture of the simulator used to explore performance under a wide range of policies.

capacity is ephemeral (by definition, not lasting to steady state) and applied to individual executions (rather than all queries).

The implementation of our simulator could simplify the specification of policies and mechanisms. Currently, the package is written in GoLang. New policies, excluding minor parameter tweaks, require new source files. Also, power budgets are checked after the simulation. This makes it hard to simulate approaches like data center sprinting that rely on the mechanisms to prevent sprinting too often.

A. Early Results

We used our simulator to compute model-based sprint ability for policies proposed in Adrenaline and ApproxHadoop [7], [9]. We extracted key features regarding the sprinting mechanism (speedup and budget) and sprinting policy in each paper. Our simulator computed the expected response time under a wide range of policies and searched for the best performance defined in the denominator of the sprint ability.

ApproxHadoop: ApproxHadoop subsamples data and drops map tasks to reduce running time for Hadoop jobs. We focused on the configuration where all data was used and map tasks were dropped. In this configuration, the price for dropping tasks is lower answer quality. Specifically, ApproxHadoop targets Hadoop jobs that output statistical answers. Answer quality is defined using the 95th percentile bound on the absolute error of the output (i.e., 1% - error). Users specify a target for answer quality and ApproxHadoop then completes each job quickly (dropping map tasks) while respecting the target quality. By default, ApproxHadoop drops map tasks randomly. We consider this policy and an alternative where ApproxHadoop drops map tasks that run slowly. We term the latter as ApproxHaddop with straggler prediction. Finally, we test the hypothesis of not using the same target answer quality for each query. Instead, we allow ApproxHadoop to sprint at a higher rates (up to 5% more dropped map tasks) as long as the average quality after 1 day is within budget. Specifically, our metric of merit is average response time across 96 jobs issued throughout the day. Table 2 shows the settings associated with our tests inspired by ApproxHadoop.

ApproxHadoop Simulation Settings		
mechanism	speedup	budget
drop maps	1,500X	avg. quality = 97.5%
policy name	query selection	worst target quality
default	random	97.5%
rand	random	95 – 97.5%
strag	stragglers	95 – 97.5%
Adrenaline Simulation Settings		
mechanism	speedup	budget
DVFS	1.57X	energy = 9.6
policy name	query selection	frequency
default	tail	15%
rand	tail	10 – 50%

Tab. 2: Sprinting mechanisms and policies simulated [7], [9].

Figure 6 plots CDF of sprint ability under two different ApproxHadoop sprinting mechanisms, namely random and straggler dropping. Each point in CDF represents a particular sprinting policy setting, i.e., frequency, measured at the simulator. Particularly, the sprint ability achieved by ApproxHadoop's default strategy is at 71%. We tested random policies that allowed various worst target quality at intervals .1%. Thanks to our simulator, we are able to efficiently explore 250 settings. And, only 74 out of 250 met the quality budget. ApproxHadoop's default strategy out performed only 3. The best policy allowed sprinting with quality loss of 4.5% for one job. As shown in the ApproxHadoop paper, dropping map requests can cause significant reduction in response time even while degrading answer quality only slightly. Comparing the random and straggler dropping policies reveals the importance of well targeted workloads on sprint ability. Straggler reduces the variance of sprint ability substantially. The median under straggler dropping policy that does not violate budget achieves over 97% sprint ability.

Adrenaline: Adrenaline exploits a DVFS mechanism capable of voltage transitions in nanoseconds. It significantly reduces energy wasted transitioning to higher frequencies. With this technology, Adrenaline sets out to speed up query executions that are likely to violate SLO. It targets (i) query executions that exceed the 85th percentile in service time and (ii) queries with total execution time within 50% of the SLO target, i.e queries whose response time exceed 500 ms. In the paper, a DVFS increase from 1.4 GHz to 2.1 GHz provide 1.57X speedup. There is a global energy budget of 30% of the baseline.

For this paper, we changed the percentile that constitutes the tail. To stay within energy budget, the change in tail was matched by a proportional change in DVFS max speed. More aggressive tail settings provided less speed up per sprint. Using data from the paper, we set the service rate to 5 ms. The utilization was 90% (labeled high in the paper). Figure 6 shows the CDF of sprint ability under any sprinting frequency explored by the simulator. One can see that sprint ability in Adrenaline had higher variance than ApproxHadoop. There are two reasons. First, our policies prohibit invalid settings. Second, Adrenaline boosts performance significantly under high utilization. Poor policies that essentially disable

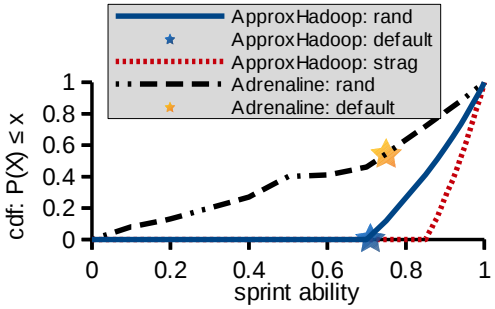

Fig. 6: Sprint ability for policies proposed in ApproxHadoop and Adrenaline.

sprinting mechanisms cause significant harm. Figure 7 shows the speedup of the Adrenaline sprinting mechanism under a fixed speedup as the tail threshold increases. This figure highlights the outsized effects under 90% utilization.

V. Related Work

Sprint ability complements metrics used to guide resource management, e.g., response time, scalability, and elasticity. This section overviews existing metrics and draws key contrasts, making the case for the research community to use sprint ability more often.

Elasticity measures how quickly a system can adapt to workload changes by provisioning and deprovisioning resouces [8]. Elasticity subsumes scale-out scalability which measures how efficiently a system can continously add resources. These metrics concern the detection of resources and distribution of work. Systems with poor elasticity and scalability are hard to model, making sprint ability hard to compute. However, elasticity presumes changes are caused by workload rather than ephemeral resources. Systems that require workload surges to scale, e.g., workloads that load balance at coarse granularities, can achieve high elasticity but poor spint ability. We expect future processors will support fine-grained sprinting within the context of a single query execution. Support for ephemeral resources that are available for very short bursts distinguishes sprint ability. Auto scaling techniques can not support short bursts yet [5], [6].

Research on powering systems with intermittent renewable energy shares a key feature with sprinting: Variability is supply side not demand side. Many recent papers propose policies to quickly adapt resources in response to solar outages [16], [18]. These systems coupled with recent papers on sprinting mechanisms have focused on proposing good policies. In contrast, sprint ability answers (1) does a sprinting mechanism permit policies that provide sufficient SLO, etc. and (2) are there policies better than the current policy.

VI. Conclusion

Sustainable computing, dark silicon and approximate computing have ushered a new era in which some processing capacity is available only as ephemeral bursts, a technique called sprinting. New techniques and metrics are needed to achieve lower response time and high throughput with sprinting mechanisms. Sprint ability complements existing

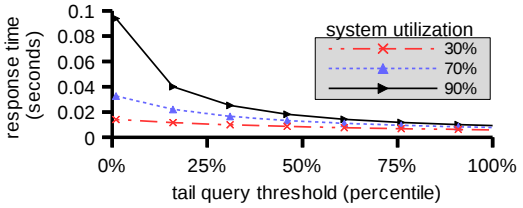

Fig. 7: Speedup from sprinting as a function of system utilization.

metrics by distinguishing the role of software policies from sprinting mechanisms. This paper argues that sprint ability, or similar metrics, should become a part of resource management vernacular. We built a simulator to start evaluating sprint ability. Early results revealed that proposed, good policies are not always best. We have open sourced our simulator to encourage others to begin studying sprint ability [14].

References

[1] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, 2014.

[2] D. J. Dean, H. Nguyen, and X. Gu. Ubl: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *IEEE ICAC*, 2012.

[3] D. J. Dean, H. Nguyen, X. Gu, H. Zhang, J. Rhee, N. Arora, and G. Jiang. Perfscope: Practical online server performance bug inference in production cloud computing infrastructures. In *ACM SOCC*, 2014.

[4] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2), 2013.

[5] A. Gandhi, S. Doroudi, M. Harchol-Balter, and A. Scheller-Wolf. Exact analysis of the m/m/k/setup class of markov chains via recursive renewal reward. In *ACM SIGMETRICS*, 2013.

[6] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang. Adaptive, model-driven autoscaling for cloud applications. In *IEEE ICAC*, 2014.

[7] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen. Approxhadoop: Bringing approximations to mapreduce frameworks. In *ASPLOS*, 2015.

[8] N. R. Herbst, S. Kounev, and R. Reussner. Elasticity in cloud computing: What it is, and what it is not. In *IEEE ICAC*, 2013.

[9] C. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. F. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *HPCA*, 2015.

[10] J. Kelley, C. Stewart, N. Morris, D. Tiwari, Y. He, and S. Elnikety. Measuring and managing answer quality for online data-intensive services. In *IEEE ICAC*, 2015.

[11] D. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *The Annals of Mathematical Statistics*, 1953.

[12] K. Lee, R. Pedarsani, and K. Ramchandran. On scheduling redundant requests with cancellation overheads. In *Allerton Conference*, 2015.

[13] R. Levin, E. Cohen, W. Corwin, F. Pollack, and W. Wulf. Policy/mechanism separation in hydra. In *ACM SIGOPS Operating Systems Review*, volume 9, 1975.

[14] Rerout Lab Git Repo. http://reroutlab.org/projects.html, 2016.

[15] E. Rotem, A. Naveh, A. Ananthakrishnan, D. Rajwan, and E. Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 2012.

[16] R. Singh, D. E. Irwin, P. J. Shenoy, and K. K. Ramakrishnan. Yank: Enabling green data centers to pull the plug. In *USENIX NSDI*, 2013.

[17] C. Stewart, K. Shen, A. Iyengar, and J. Yin. Entomomodel: Understanding and avoiding performance anomaly manifestations (winner of best paper award). In *IEEE MASCOTS*, 2010.

[18] Z. Xu, N. Deng, C. Stewart, and X. Wang. Cadre: Carbon-aware data replication for geo-diverse services. In *IEEE ICAC*, 2015.

[19] W. Zheng and X. Wang. Data center sprinting: Enabling computational sprinting at the data center level. In *IEEE ICDCS*, 2015.