To Migrate or to Wait: Bandwidth-Latency Tradeoff in Opportunistic Scheduling of Parallel Tasks

Ting He*, Shiyao Chen†, Hyoil Kim‡, Lang Tong†, and Kang-Won Lee*
*IBM T.J. Watson Research Center, Yorktown, NY, USA. Email: {the,kangwon}@us.ibm.com
†School of ECE, Cornell University, Ithaca, NY, USA. Email: {sc933,lt35}@cornell.edu
‡School of ECE, Ulsan National Institute of Science and Technology, Korea. Email: hkim@unist.ac.kr

***Abstract*—We consider the problem of scheduling low-priority tasks onto resources already assigned to high-priority tasks. Due to burstiness of the high-priority workloads, the resources can be temporarily underutilized and made available to the low-priority tasks. The increased level of utilization comes at a cost to the low-priority tasks due to intermittent resource availability. Focusing on two major costs, bandwidth cost associated with migrating tasks and latency cost associated with suspending tasks, we aim at developing online scheduling policies achieving the optimal bandwidth-latency tradeoff for parallel low-priority tasks with synchronization requirements. Under Markovian resource availability models, we formulate the problem as a Markov Decision Process (MDP) whose solution gives the optimal scheduling policy. Furthermore, we discover structures of the problem in the special case of homogeneous availability patterns that enable a simple threshold-based policy that is provably optimal. We validate the efficacy of the proposed policies by trace-driven simulations.**

I. Introduction

Cloud-based service model often promises to provide infinite scalability in satisfying computing requests on demand. To guarantee the satisfaction of their service level agreement (SLA) with clients, the providers are forced to plan their physical capacity for the peak load, thus leaving much of the resource under-utilized during normal periods of operation.

A common remedy is to perform "valley filling", i.e., filling the under-utilized periods left by *frontend tasks* (e.g., web applications) with *backend tasks* (e.g., high performance computing applications) that can be suspended in case the frontend loads increase. Such overlaying resource allocation can improve resource utilization without sacrificing the SLA for frontend tasks. The drawback is that the performance of the backend tasks cannot be guaranteed since they only have intermittent access to resources, especially for synchronized parallel tasks. Synchronized parallel tasks abound in scientific computing, Pregel [1], and MapReduce [2] applications, where the data dependency between computation steps requires the tasks to progress almost simultaneously. In the sequel, we will only focus on backend tasks and simply refer to them as tasks.

Advances in virtualization technology have provided tools to manage these tasks more intelligently. In addition to suspension of tasks during execution (via VM suspension), most hypervisors also support live migration of tasks with negligible ($\sim$ seconds) interruption in their execution. This means that instead of waiting indefinitely at one server during busy periods, we may choose to migrate the affected backend task to another server with lighter loads. The savings in waiting time come at the cost of significant traffic loads on the cloud network due to high volumes of data/code transfer, which in turn adds to the operation cost of the cloud. Given a pool of intermittently available resources, what is the optimal way of scheduling suspendable/migratable tasks onto them? How do we trade off the bandwidth cost due to migration and the latency cost due to suspension? And how do the answers depend on the availability behaviors of these resources? These are the main questions we seek to answer in this paper.

This work was supported by the U.S. National Institute of Standards and Technology under Agreement Number 60NANB10D003.

A. Related Work

The problem of resource sharing has been a main theme of computer scheduling, where techniques such as cycle stealing [3] have been proposed to transparently share computing resources among jobs of different priorities. Being non-intrusive to high-priority jobs, these techniques postpone low-priority jobs to avoid conflicts, and many efforts have since been made to minimize such sacrifice.

Existing work can be divided into three categories: (i) *complete control*, where both types of jobs are scheduled jointly [4], (ii) *one-sided control with complete knowledge*, where the scheduler of low-priority jobs faces fluctuating resources but has full knowledge of the fluctuations throughout time [5], and (iii) *one-sided control with partial knowledge*, as in this paper, where the knowledge of resource fluctuations is limited to past observations and long-term statistics. The last case is most generally applicable due to its non-intrusiveness and minimalistic assumption, but it also suffers the most performance loss. Approaches to improving the performance include building sophisticated resource models [6] and leveraging advanced tools. Migration, in particular, is a powerful tool provided by virtualization that allows the movement of jobs at run time. It is, however, a costly operation that must be used with caution to balance job performance and operation cost.

A related problem is *scheduling with switching cost*, where a server is scheduled among multiple jobs. As an extension of Multi-Armed Bandits (MAB), this problem has received much attention due to its broad applicability [7]. Specific results include index-based policies to handle switching costs/delays and

978-1-4673-0775-8/12/$31.00 ©2012 IEEE

precedence relations; see [8] and references therein. Our problem is a dual in that we schedule tasks among multiple servers, with three crucial differences: (a) our scheduler does not have to fully utilize all the servers; (b) while unattended jobs remain the same, unutilized servers can change states due to frontend activities; (c) in contrast to independent jobs in previous work, we consider parallel tasks that must run simultaneously.

B. Summary of Results

We consider the opportunistic scheduling of parallel tasks onto stochastically available servers, focusing on the tradeoff between bandwidth (migration) and latency (waiting). We unify the two metrics into a single cost that can capture a range of tradeoff requirements. Our goal is to develop online scheduling policies that can minimize the total cost. Our main contributions include the following:

Optimal solution based on MDP: Under the assumption that server availability processes can be modeled as independent ON/OFF Markov chains, we show that the optimal policy is the solution to an MDP, which has exponential complexity in the number of servers.

Closed-form optimal solution in a special case: We develop an efficient, closed-form policy that guarantees optimality in the case when all servers behave homogeneously (with i.i.d. availability processes) and there are always sufficient available servers to run the tasks. The policy is a threshold policy that will wait (migrate) if the number of available servers among those currently hosting tasks is below (above) a threshold, where the threshold is determined by the per-migration cost and a precomputed partition of the cost range.

Evaluation on traces: We evaluate the proposed policies on traces from a real data center. The simulations verify that the threshold policy is near optimal in near-homogeneous cases. Compared with a heuristic solution (myopic policy), the threshold policy is provably optimal when the servers exhibit homogeneous availability without increasing the complexity, although we observe that the two policies can perform similarly as the number of parallel tasks increases.

The rest of the paper is organized as follows. Section II formulates the problem. Section III presents our scheduling policies. Section IV shows the evaluation results. Then Section V concludes the paper.

II. Problem Formulation

We will focus on the scheduling of a single set of parallel tasks. Our solution can be applied repeatedly to schedule multiple sets of tasks, although it is beyond our scope to study the order of scheduling.

A. Workload Modeling

We model a set of parallel tasks as a tuple (n, τ), where n is the number of tasks, each running on a separate server (or VM), and τ the number of time slots required per task, assuming discrete time. We assume strong synchronization where the tasks must run simultaneously, i.e., all the n tasks have to be suspended if any server hosting one of them becomes unavailable. In practice, a weaker requirement may suffice, but we focus on strong synchronization to ensure that the resulting schedule is feasible under arbitrary synchronization requirements.

B. Server Resource Modeling

Given a pool of $N > n$ servers, each with time-varying available resources depending on the frontend workloads, we model their capability of hosting the backend tasks by binary ON/OFF processes, where a server is ON if and only if its available resource satisfies the requirement of a backend task. Without loss of generality, we assume each server can host at most one task; in cases when a physical server can host multiple tasks, we predivide servers into task-sized partitions and simply refer to these partitions as "servers". We refer to a server as a "host" if it is selected to host a task. Note that a host can be either ON or OFF. Denote the server availability processes by $\{\mathbf{a}_t\}_{t=1}^{\infty}$, where $\mathbf{a}_t = (a_{h,t})_{h=1}^{N} \in \{0,\ 1\}^N$ are the indicators for each server h to be ON at time t.

C. Bandwidth-Latency-Optimized Scheduling

We want to schedule the parallel tasks onto the pool of intermittently available servers so that they can be finished with minimum bandwidth-latency cost. We measure the *bandwidth cost* by the number of migrations m (i.e., the total number of times that each of the tasks is migrated) in a linear form: $c_m(m) = \gamma m$, where $\gamma \geq 0$ denotes the cost per migration due to the communication load for data/code transfer. The *latency cost* is measured by the total waiting time w that the tasks spend waiting for available resources from arrival until completion, again assumed to be linear: $c_w(w) = w$ (the scaling factor is incorporated into γ). We hasten to note that our approach can handle more complicated cost functions including those modeling SLA penalties; details are omitted due to space limit. The total cost is thus $c(m,\ w) = w + \gamma m$. We say that a scheduling policy achieves the *optimal bandwidth-latency tradeoff* if it minimizes the total cost $c(m,\ w)$.

The inherent tradeoff between bandwidth and latency implies that the scheduler needs to decide how aggressively it should migrate tasks to achieve the desirable tradeoff, controlled by the parameter γ. For example, the extreme values $\gamma = 0$ and $\gamma = \infty$ will lead to no waiting or no migration scheduling. In general, we have the following qualitative relationship. All the proofs in this paper are given in [9].

Proposition II.1. *Under the optimal scheduling policy, the number of migrations m decreases and the waiting time w increases monotonically with the increase of γ.*

Here γ is a system-dependent parameter that depends on the task size, the cloud network capability, the penalty for delayed completion, etc. In the sequel, we will assume γ is given and focus on developing the cost-optimal scheduling policy.

III. Scheduling Policies

In each slot, the scheduler faces multiple decisions: if some of the current hosts become unavailable, should it suspend all the tasks and wait for the hosts to be available again, or

should it migrate the interrupted tasks to other available servers so that the tasks continue running? In the case of migration, which servers should the tasks migrate to? The answers to these questions clearly depend on the *future* availability behaviors of the hosts and the candidate servers. Since such information is generally unavailable, the scheduler has to make decisions based on the availability *history*, leading to *online scheduling*. In this section, we propose a simple statistical model for server behaviors, based on which we develop an optimal solution for the general case and an efficient, closed-form solution for a special case with practical relevance.

A. Server Availability as ON/OFF Markov Chains

We examine the utilization traces of a real data center to extract statistical models of server behaviors. Using processor time as the key resource, we quantize the fractional utilization traces by a given threshold ζ to generate ON/OFF availability traces (ON if utilization $\leq \zeta$). Parameter fitting shows that the lengths of ON/OFF intervals roughly follow the Geometric distribution, as illustrated in Fig. 1 (a). This motivates us to model the server availability processes by ON/OFF Markov chains, whose ON/OFF intervals are exactly geometrically distributed. For simplicity, we assume the Markov chains to be independent across servers, and the transition probabilities $(p_{ij}^h)_{i,j\in\{0,1\}}$ for each server $h = 1, \ldots, N$ are known. This model does incur some error; see Fig. 1 (b) for the distribution of the error as measured by the Kolmogorov-Smirnov statistic[1]. Using this model as an approximation allows us to develop efficient scheduling policies that capture temporal dependencies in server availability. Our later evaluations show that this approximation has minimal impact on scheduling performance (see Section IV).

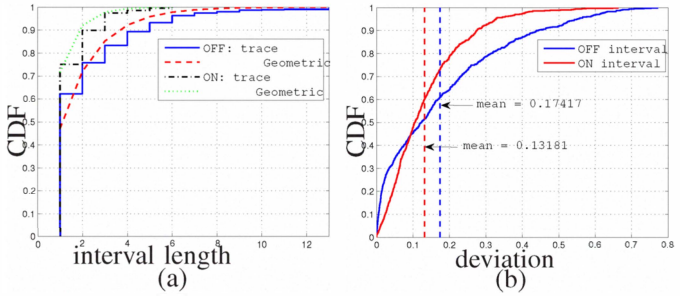

Fig. 1. Geometric fitting of ON/OFF intervals: (a) fitting for a single server; (b) distribution of fitting error over all servers ($\zeta = 15\%$).

B. A General Solution by Markov Decision Process

To achieve the optimal tradeoff, each migration decision needs to take into account a window of future server availability. This need of lookahead naturally leads to a dynamic programming approach, where the problem can be formulated as a Markov Decision Process (MDP) [10] as follows.

The MDP evolves slot by slot on a joint state $(\tau_t, \mathbf{x}_t, \mathbf{a}_t)$, where τ_t is the residual workload ($\tau_0 = \tau$), $\mathbf{x}_t$ the current hosts, and $\mathbf{a}_t$ the states of all the servers, all at slot t. Here $\mathbf{x}_t = (x_{h,t})_{h=1}^N$, where $x_{h,t}$ is an indicator that server h is selected as a host at time t, with values in $\mathcal{X} \triangleq \{\mathbf{x} \in \{0,1\}^N : \sum_{h=1}^N x_h = n\}$. Let $C(\tau_t, \mathbf{x}_t, \mathbf{a}_t)$ denote the minimum expected cost starting from state $(\tau_t, \mathbf{x}_t, \mathbf{a}_t)$. The optimal schedule is the solution to the following recursion:

$$C(\tau_t, \mathbf{x}_t, \mathbf{a}_t) = \min_{\mathbf{x}\in\mathcal{X}} \Big[w(\mathbf{a}_t, \mathbf{x}) + \gamma m(\mathbf{x}_t, \mathbf{x}) + \mathbb{E}[C(\tau_{t+1}, \mathbf{x}_{t+1}, \mathbf{a}_{t+1})] \Big], \quad (1)$$

where $w(\mathbf{a}_t, \mathbf{x}) \triangleq 1 - \lfloor \frac{1}{n} \sum_{h=1}^N x_h a_{h,t} \rfloor$ is the waiting indicator, and $m(\mathbf{x}_t, \mathbf{x}) \triangleq \frac{1}{2} \sum_{h=1}^N |x_{h,t} - x_h|$ the number of migrations to change hosts from $\mathbf{x}_t$ to $\mathbf{x}$. Note that the $\lfloor \cdot \rfloor$ operator reflects the synchronization requirement, as $w(\mathbf{a}_t, \mathbf{x}) = 0$ only if all the selected hosts (i.e., $x_h = 1$) are ON (i.e., $a_{h,t} = 1$). Under action $\mathbf{x}$, the state transits to $\tau_{t+1} = \tau_t - 1 + w_{t+1} - w_t$ and $\mathbf{x}_{t+1} = \mathbf{x}$, and the expectation is over the next server state $\mathbf{a}_{t+1}$. The boundary condition is $C(0, \mathbf{x}_t, \mathbf{a}_t) = 0$. This formulation implies a finite state space of size $\tau\binom{N}{n}2^N$, and can be solved by standard MDP solvers.

A case of particular interest is that of *long-lasting tasks* ($\tau \gg 1$), where the problem can be simplified by focusing on the long-term cost rate. To minimize the *discounted cost rate* with discount factor $\beta \in (0, 1)$, the optimal schedule is given by:

$$C(\mathbf{x}_t, \mathbf{a}_t) = \min_{\mathbf{x}\in\mathcal{X}} \Big[w(\mathbf{a}_t, \mathbf{x}) + \gamma m(\mathbf{x}_t, \mathbf{x}) + \beta \mathbb{E}[C(\mathbf{x}_{t+1}, \mathbf{a}_{t+1})] \Big], \quad (2)$$

starting from $\mathbf{x}_1 = \arg\min_{\mathbf{x}\in\mathcal{X}} C(\mathbf{x}, \mathbf{a}_1)$. Average cost rate can be addressed analogously. The size of its state space is now reduced to $\binom{N}{n}2^N$.

Remark: Unfortunately, even in the simpler case (2), the complexity of the MDP is still exponential in N. One remedy is to use a suboptimal policy, e.g., the *myopic policy*. The myopic policy greedily minimizes the immediate cost $w(\mathbf{a}_t, \mathbf{x}) + \gamma m(\mathbf{x}_t, \mathbf{x})$, yielding a threshold decision: if the number of hosts that are OFF is less than $1/\gamma$ and there are enough ON servers, migrate all the tasks on OFF hosts to unused ON servers; otherwise, suspend the tasks and wait. This policy can be substantially suboptimal, which motivates us to explore efficient policies with better performance as presented next.

C. Closed-Form Solution for Homogeneous Servers

Consider the case of long-lasting tasks (2). We resolve the complexity issue in the special case where the server availability processes are i.i.d. across servers, and there are always enough (i.e., $\geq n$) ON servers. This case usually occurs for a pool of servers running the same frontend application, where the load balancer of the application naturally induces i.i.d. loads. In this section, we derive the optimal policy in closed form based on a sequence of simplifications.

Our first simplification is by noting that it is sufficient to only remember the number of ON hosts, i.e., the servers currently hosting tasks and being ON, at each slot. This is because we do not need to remember which servers are selected as hosts

[1]Only servers with enough (≥ 50) ON/OFF intervals are considered.

since all servers in the same state are identical. Denote by s'_t the number of ON hosts at slot t *before* migration, and s_t the same *after* migration. Note that s_t may be different from s'_t as the migration action may change which servers serve as hosts. We can represent an arbitrary (stationary, randomized) scheduling policy by a matrix $\mathbf{M} = (M_{ij})_{i,j=0}^{n}$, where $M_{ij} = \Pr\{s_t = j | s'_t = i\}$ is the probability that the policy will modify the number of ON hosts from i to j by migration, satisfying $\sum_{j=0}^{n} M_{ij} = 1$ for all i. Let $\mathbf{P} = (P_{ij})_{i,j=0}^{n}$ denote the transition probability of the number of ON hosts *without* migration: $P_{ij} \triangleq \Pr\{s'_t = j | s_{t-1} = i\}$. Using the single-server transition probabilities $(p_{ij})_{i,j\in\{0,1\}}$, we can compute $\mathbf{P}$ by

$$P_{ij} = \sum_{k=(j-n+i)_+}^{\min(i,j)} \binom{i}{k} p_{11}^{k} p_{10}^{i-k} \binom{n-i}{j-k} p_{01}^{j-k} p_{00}^{n-i-j+k}. \quad (3)$$

Define $\mathbf{Q} = (Q_{ij})_{i,j=0}^{n}$, with $Q_{ij} \triangleq |i-j|$ being the minimum number of migrations to change the number of ON hosts from i to j. Denote the initial distribution by $\boldsymbol{\lambda} = (\lambda_i)_{i=0}^{n}$ with $\lambda_i = \Pr\{s'_1 = i\}$.

The computation of the optimal policy now becomes $\mathbf{M}^* = \arg\max_{\mathbf{M}} R_{\mathbf{M}}(\gamma)$ for[2]

$$R_{\mathbf{M}}(\gamma) \triangleq \pi_n - \gamma(\boldsymbol{\lambda}^T + \beta \boldsymbol{\pi}^T \mathbf{P}) \cdot \mathrm{diag}(\mathbf{MQ}), \quad (4)$$

where $\boldsymbol{\pi}^T = \boldsymbol{\lambda}^T \mathbf{M}(\mathbf{I} - \beta \mathbf{P}\mathbf{M})^{-1}$. Here $\boldsymbol{\pi} = (\pi_i)_{i=0}^{n}$ denotes the discounted long-term distribution of s_t, defined as $\pi_i = \mathbb{E}[\sum_{t=1}^{\infty} \beta^{t-1} \mathbb{1}_{s_t=i}]$. It has to satisfy the equilibrium equation $\pi_j = \sum_{i=0}^{n} \lambda_i M_{ij} + \beta \sum_{i,k=0}^{n} \pi_i P_{ik} M_{kj}$, which is used to compute $\boldsymbol{\pi}$. The objective function $R_{\mathbf{M}}(\gamma)$, called the *total reward*, represents the complement of the total cost, which is given by $1/(1-\beta) - R_{\mathbf{M}}(\gamma)$. Here π_n is the expected discounted task processing time, capturing the synchronization requirement since only the time that all the n hosts are ON is considered, and $(\boldsymbol{\lambda}^T + \beta \boldsymbol{\pi}^T \mathbf{P}) \cdot \mathrm{diag}(\mathbf{MQ})$ is the expected discounted total number of migrations. This formulation reduces the MDP to an $O(n^2)$-variable optimization over $\mathbf{M}$.

Our second simplification is due to the following fact.

Lemma III.1. *There is no need to migrate any task in a slot in which the tasks will wait.*

This lemma implies that the optimal policy should only migrate tasks if they will run as a result. Thus, it suffices to restrict $\mathbf{M}$ to the form: $M_{ii} = p_i$, $M_{in} = 1 - p_i$, and $M_{ij} = 0$ otherwise for $i = 0, \ldots, n-1$ ($M_{nn} = 1$). Moreover, it is known that randomization does not improve performance of MDP [10], which means $p_i \in \{0, 1\}$, i.e., the optimal policy will either "wait" (if $p_i = 1$) or "migrate" (if $p_i = 0$) at a given state $s'_t = i$, with a finite number (2^n) of possibilities.

Our last simplification is due to the following property.

Lemma III.2. *The optimal policy is a* threshold policy $\mathcal{T}_i$ *for some $i \in \{0, \ldots, n\}$, which will wait if $s'_t < i$ and migrate otherwise (including no migration if $s'_t = n$).*

[2]Here $\mathrm{diag}(\mathbf{A})$ denotes the diagonal of matrix $\mathbf{A}$, and $\mathbf{I}$ the identity matrix. All vectors are column vectors.

This lemma simplifies the solution space from all the 2^n candidate policies to the $n+1$ threshold policies $\mathcal{T}_0, \ldots, \mathcal{T}_n$. Let $\mathcal{P}_i \triangleq \pi_n|_{\mathcal{T}_i}$ and $\mathcal{M}_i \triangleq (\boldsymbol{\lambda}^T + \beta \boldsymbol{\pi}^T \mathbf{P})\mathrm{diag}(\mathbf{MQ})|_{\mathcal{T}_i}$ be the processing and the migration metrics under policy $\mathcal{T}_i$ ($i = 0, \ldots, n$). Then they satisfy the following order.

Lemma III.3. *Among the threshold policies, we have $\mathcal{P}_0 \geq \mathcal{P}_1 \geq \ldots \geq \mathcal{P}_n$ and $\mathcal{M}_0 \geq \mathcal{M}_1 \geq \ldots \geq \mathcal{M}_n$.*

Together, these simplifications specify the optimal policy in closed form.

Theorem III.4. *There exist thresholds $\gamma_i \triangleq (\mathcal{P}_i - \mathcal{P}_{i-1})/(\mathcal{M}_i - \mathcal{M}_{i-1})$ ($i = 1, \ldots, n$) such that the threshold policy $\mathcal{T}_i$ is optimal if and only if $\gamma \in [\gamma_i, \gamma_{i+1}]$ ($\gamma_0 \triangleq 0$, $\gamma_{n+1} \triangleq \infty$).*

Proof: Since $\mathcal{T}_0$ is optimal at $\gamma = 0$ and $\mathcal{T}_n$ optimal at $\gamma = \infty$, the ordering of threshold policies together with Proposition II.1 implies that the optimal policy is $\mathcal{T}_0, \ldots, \mathcal{T}_n$ in this order as γ increases. Given the reward $R_i(\gamma)$ for $\mathcal{T}_i$ as defined in (4), it is easy to see that the transition between policies must occur at the intersecting points between $R_i(\gamma)$'s. Since $R_i(\gamma) = \mathcal{P}_i - \gamma \mathcal{M}_i$, the intersecting point between $\mathcal{T}_{i-1}$ and $\mathcal{T}_i$ is given by $\gamma_i = (\mathcal{P}_i - \mathcal{P}_{i-1})/(\mathcal{M}_i - \mathcal{M}_{i-1})$ ($i = 1, \ldots, n$). ∎

The final result gives a fully closed-form, optimal solution based on a partition of the range of γ, as illustrated in Fig. 2 (a), which maps each interval of the partition to one of the threshold policies (boundary point γ_i is mapped to either $\mathcal{T}_{i-1}$ or $\mathcal{T}_i$). Compared with the exponential-complexity MDP solution in Section III-B, this solution significantly reduces the complexity to (pseudo) constant in N ($O(N)$ in general) since it only needs to examine the states of the n hosts and find $k \leq n$ replacement hosts in the case of migration.

Recall that the myopic policy also has a threshold structure (Section III-B). In particular, it leads to a partition on γ by thresholds $\gamma_i^{\text{MY}} = 1/(n-i+1)$ ($i = 1, \ldots, n$) and $\gamma_0^{\text{MY}} \triangleq 0$, $\gamma_{n+1}^{\text{MY}} \triangleq \infty$ such that the myopic policy is reduced to the threshold policy $\mathcal{T}_i$ for $\gamma \in [\gamma_i^{\text{MY}}, \gamma_{i+1}^{\text{MY}}]$, as illustrated in Fig. 2 (b). We observe that γ_i^{MY} converges to γ_i as p_{01} and p_{11} converge and diverges otherwise, consistent with the fact that myopic policy is optimal if server states are i.i.d. both across servers and over time.

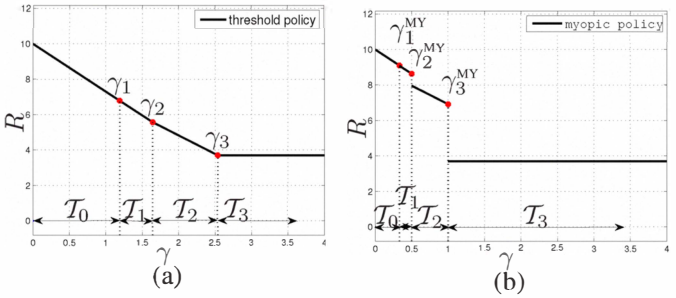

Fig. 2. Threshold structure of the optimal policy (a) and the myopic policy (b) ($n = 3$, $\beta = 0.9$, $p_{01} = 0.1$, $p_{11} = 0.9$, $\boldsymbol{\lambda} = (0, 0, 0, 1)^T$).

Remark: Although we have assumed the system to always have enough ON servers, our solution applies naturally to the

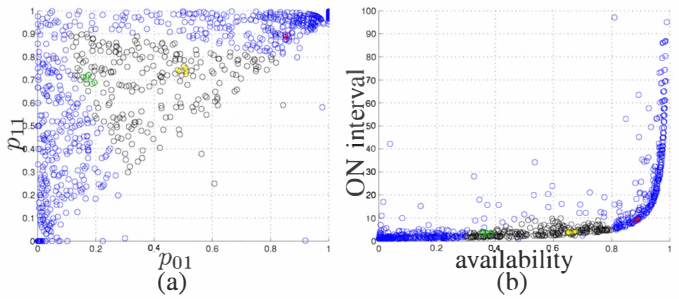

Fig. 3. Selected servers: totally 232 servers (○); near-homogeneous subsets of servers with low (□), medium (◇), and high (+) availability, 6 each.

other cases, with the modification that the policy will only migrate if the total number of ON servers is at least n.

IV. Performance Evaluation

We evaluate the proposed policies by simulating them on availability traces derived from real measurements. We collect CPU utilization traces from a pool of servers hosting business users at 15-minute intervals and convert them into availability traces using a threshold (see Section III-A; $\zeta = 15\%$). All the simulations last for at least 1000 slots (only traces $\geq$ 1000 slots are considered). Assume $\beta = 0.999$.

To test homogeneous scenarios, we select three representative server sets with low, medium, and high availability, respectively, as illustrated in Fig. 3, keeping the servers in each set relatively homogeneous. To test heterogeneous scenarios, we randomly select N servers from a larger pool of heterogeneous servers. Since many traces are degenerate, we filter out servers with too good/bad availability or too few ON/OFF fluctuations, as shown in Fig. 3, to focus on the dynamic cases.

Due to the high complexity of MDP, the optimal policy in the general case can only be computed for small N and n. Thus, we perform a small-scale simulation with $n = 2$ and $N = 6$ to compare the alternative policies, the threshold policy and the myopic policy, with the optimal. We compare the overall costs as γ increases for selected servers with various availability[3] and heterogeneity; see Fig. 4. All the policies are able to adapt to the change of γ. The threshold policy shows near-optimal performance in the near-homogeneous case (Fig. 4 (a)), but incurs performance loss at large γ in the heterogeneous case (Fig. 4 (b)), mostly because of its random selection of new hosts (since all the ON servers are considered identical). The myopic policy shows a similar trend because it also has a threshold structure (see Fig. 2), although it performs worse than the threshold policy, especially in the heterogeneous case.

We have also increased the scale of the simulation to $n = 10$ and $N = 100$, and evaluate only the threshold and the myopic policies; see Fig. 5. We see that the two policies perform similarly when averaged over different sets of N servers (Fig. 5 (a)), with slightly better performance for the threshold policy on some server sets (Fig. 5 (b)). While this observation suggests comparable asymptotic performance for the myopic

[3]Fig. 4 (a) is for servers with low availability; see [9] for the cases of medium/high availability.

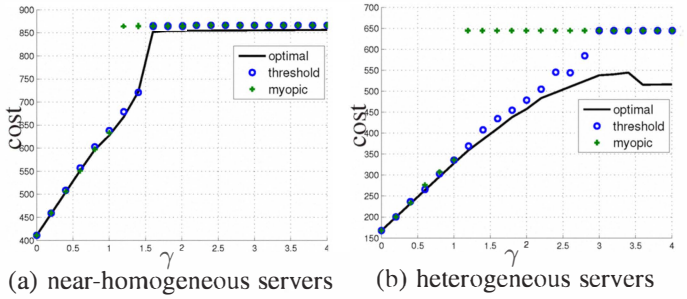

Fig. 4. Small-scale simulations.

policy, we point out that the threshold policy has the same complexity and is thus still preferable due to its guaranteed optimality.

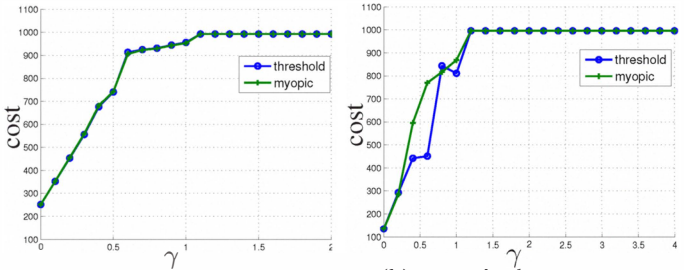

Fig. 5. Larger scale simulations.

We repeat these simulations on synthetic availability processes generated by Markov chains parameterized according to the traces. The synthetic simulations yield observations similar to the above, which validates our availability model; see [9].

V. Conclusion

We study the problem of opportunistic scheduling of parallel backend tasks with focus on the tradeoff between migration and waiting. Although it is generally hard to compute the optimal policy for large server pools, we give an efficient threshold policy that is provably optimal for homogeneous servers.

References

[1] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A System for Large-Scale Graph Processing," in *PODC*, 2009.

[2] "Introduction to Parallel Programming and MapReduce," http://code.google.com/edu/parallel/mapreduce-tutorial.html.

[3] K. D. Ryu and J. K. Hollingsworth, "Unobtrusiveness and Efficiency in Idle Cycle Stealing for PC Grids," in *IPDPS*, 2004.

[4] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade, "Enabling Resource Sharnig between Transactional and Batch Workloads Using Dynamic Application Placement," in *MIDDLEWARE*, 2008.

[5] V. T. Chakaravarthy, V. Pandit, Y. Sabharwal, and D. P. Seetharam, "Varying Bandwidth Resource Allocation Problem with Bag Constraints," in *IPDPS*, 2010.

[6] B. Favadi, D. Kondo, J.-M. Vincent, and D. P. Anderson, "Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home," *Trans. PDS*, 2011.

[7] T. Jun, "A Survey on the Bandit Problem with Switching Costs," *De Economist*, 2004.

[8] K. D. Glazebrook and D. Ruiz-Hernandez, "A Restless Bandit Approach to Stochastic Scheduling Problems with Switching Costs," 2005, preprint.

[9] T. He, S. Chen, H. Kim, K.-W. Lee, and L. Tong, "Bandwidth-Latency Tradeoff in Opportunistic Task Scheduling: Supporting Materials," IBM, Tech. Rep., 2011, http://researcher.ibm.com/files/us-the/rc071101.pdf.

[10] M. L. Puterman, *Markov Decision Processes.* Wiley, 1994.