H-0296 (H1101-002) January 5, 2011
Computer Science

IBM Research Report

Scheduling Live Migrations of Virtual Machines during Host Evacuation

Alex Glikson, Amir Epstein, Assaf Israel, John Marberg
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Scheduling Live Migrations of Virtual Machines During Host Evacuation

Alex Glikson, Amir Epstein, Assaf Israel, John Marberg
IBM Haifa Research Lab, Haifa 31905, Israel
{glikson,amire,assafi,marberg}@il.ibm.com

ABSTRACT

Server virtualization introduces new capabilities that improve the efficiency of data centers. One such capability is live migration of virtual machines – the ability to move an operational virtual machine from one physical host to another without perceivable downtime of applications running within the virtual machine. Efficient management of live migrations is particularly important in scenarios involving a large number of migrations, such as host maintenance or workload consolidation, when all virtual machines running on a host need to be evacuated to other hosts. Since live migration is demanding on host and network resources, decreasing the time needed for host evacuation can improve system availability and reduce administrative costs. In an emergency, when a host is likely to become unavailable quickly, minimizing the duration of host evacuation narrows the risk of losing applications.

This paper presents the problem of scheduling virtual machine migrations during host evacuation. Given a set of migrations emanating from a single host, each migration having a length and a predefined destination host, and a set of limits on the number of migrations each host can handle concurrently, the goal is to obtain a migration schedule whose total length (makespan) is minimal. The problem is known to be *NP-hard*. We obtain approximation bounds on the makespan for two common greedy scheduling heuristics: *LS* and *LPT*, and show that an optimal schedule can be generated using *LS*. We then devise a $(2+\epsilon)$-approximation scheme for the problem. Finally, we introduce a family of custom heuristics, and demonstrate empirically that they outperform the common heuristics. We also demonstrate that the custom heuristics are comparable with a scheduler deploying a commercial constraint programming solver, while our heuristics are considerably simpler and faster.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods, Scheduling*; K.6.4 [**Management of Computing and Information Systems**]: System Management

General Terms

Theory, Algorithms

Keywords

virtual machines, live migration, scheduling, makespan, approximation

1. INTRODUCTION

Live migration is the process of moving a Virtual Machine (VM) from one host (physical machine) to another, without noticeable downtime or disruption to the software stack running within the VM. Live migration enables new capabilities in management of virtualized workloads, such as non-disruptive maintenance and load balancing. The migration process itself is non-trivial, and requires significant amount of CPU resources and network bandwidth, in order to transfer the state of the operational VM (dominated by the content of RAM allocated to the VM) from the source host to the destination host [4].

Moreover, in typical implementations of live migration the state continues to change during the migration itself (e.g., the application continues to write to RAM). It is important to ensure that sufficient resources are allocated to the migration process. Otherwise, migration could take a very long time, and the downtime might be disruptive.

Efficient planning and management of live migration is particularly important in scenarios involving a large number of migrations that need to happen within a small period of time. In such case, it is desirable to schedule migrations efficiently, taking into account attributes such as configuration of the VMs (e.g., memory size), resources at the hosts (e.g., number of network adapters), and limitations in the communication media (e.g., network bandwidth). Reducing the overall time to complete all migrations will typically improve system availability and reduce administrative costs.

The main scenario involving large number of migrations addressed in this paper is *host evacuation* – migrating all VMs residing in a particular physical host. Host evacuation is applied before planned or emergency host maintenance, and during workload consolidation. In an emergency, when the

period of time until the host crashes or or is forced to shut down is limited, minimizing the duration of host evacuation narrows the risk of losing applications.

1.1 Problem Definition

We now present a model for the problem of scheduling migrations of virtual machines. We then focus particularly on the case of host evacuation. The objective is to schedule the migrations concurrently under constraints imposed by the model, such that the overall time to complete all migrations is as small as possible.

Our model is equivalent to the model introduced by Coffman, Garey, Johnson and LaPaugh [6] for the problem of scheduling file transfers among nodes in a network (which, in fact, is a problem of similar nature). It is defined as follows.

- The topology is a directed multigraph $G(H, M)$, where H is the set of vertices, representing the hosts, and M is a set of directed edges representing the migrations.
- Each migration $m_i \in M$ is characterized by a *source* host $s(m_i) \in H$ and *destination* host $d(m_i) \in H$, and by a *length* $l(m_i) > 0$.
- Each host $h_j \in H$ has a *host concurrency limit*, $c(h_j)$, defined as the maximum number of migrations that can run concurrently, for which h_j is either source or destination.
- Migrations proceed directly between the corresponding source and destination hosts, without routing through intermediate hosts, and are non-preemptive.

A *schedule* $S(M)$ is an assignment of a start time $t(m_i)$ to every migration $m_i \in M$, such that all concurrency limits are complied with at all times. The smallest start time is assumed to be 0. The total length of the schedule, called *makespan*, is $L(S(M)) = \max_i (t(m_i) + l(m_i))$. A schedule is *optimal* if its makespan is the smallest among all possible schedules for the given problem instance. The optimal makespan is denoted $OPT(M)$.

Host evacuation scheduling is modeled as a private case of migration scheduling. The topology is a single level directed tree multigraph, with the evacuated host as root (*star* topology). We denote the evacuated host h_0. Thus, for any migration m_i, $s(m_i) = h_0$. Without loss of generality, we assume that for any host h_j, $c(h_j) \leq c(h_0)$ (scheduling cannot exceed the concurrency limit of either source or destination).

Let $C(H) = \min (c(h_0), \sum_{j \neq 0} c(h_j))$. Clearly, at most $C(H)$ migrations can proceed concurrently at any time during host evacuation. Hence, we call $C(H)$ the *effective concurrency*. Although concurrency limits may vary among hosts, typically the evacuated host is likely to be the main bottleneck affecting concurrency.

When a generic problem instance is discussed, or if the specific problem instance is implied by the context, we denote the effective concurrency C, the optimal makespan OPT, the schedule S, and its makespan $L(S)$. If the schedule is implied, we denote the makespan L.

Migration scheduling is *NP*-hard, even when restricted to host evacuation. This is shown in [6], by reduction from the multiprocessor scheduling problem to a two-node file transfer scheduling problem. Scheduling heuristics can be applied, with the goal of finding a schedule whose makespan is as close as possible to the optimal. Bounds on the approximation ratio L/OPT for various heuristics and topologies are of interest.

1.2 Our Contributions

This work focuses on polynomial time approximation heuristics for host evacuation, aproblem known to be *NP*-hard. We examine two common greedy scheduling heuristics: List Scheduling (LS), and Longest Processing Time First (LPT), and present several new heuristics.

First, we prove that for host evacuation, an optimal schedule can always be generated using the LS algorithm. This is in contrast to the general topology case, where for some instances the optimal schedule is inherently non-greedy, as shown in [6].

Subsequently, we derive approximation bounds on the makespan of the LS and LPT heuristics for the host evacuation topology. These bounds are better than the corresponding bounds for the general topology given in [6]. For LS, we show a tight approximation ratio of $3 - \frac{2}{\sqrt{C}}$, compared to $3 - \frac{2}{C}$ for arbitrary topology. For LPT, we show an upper bound of $\frac{7}{3} - \frac{2}{\sqrt{3C}}$, compared to $\frac{5}{2} - \frac{1}{C}$ in the general case.

We also present a polynomial time $(2+\epsilon)$-approximation algorithm for the host evacuation problem, leveraging a Polynomial Time Approximation Scheme (PTAS) of the classical multiprocessor scheduling problem.

Finally, we introduce a family of custom-designed scheduling heuristics, tailored specifically for host evacuation, and demonstrate empirically that their results outperform the common heuristics. Also, having implemented a scheduler which deploys a commercial constraint programming solver [16], we demonstrate that the custom schedulers and the solver achieve comparable results, whereas the custom heuristics seem to be significantly more efficient and have simpler implementation.

1.3 Paper Organization

The rest of the paper is organized as follows. In Section 2 we survey related work. Section 3 discusses approximation algorithms for host evacuation. Two common greedy scheduling heuristics: LS and LPT, are analyzed in terms of approximation bounds on the makespan. We also present a polynomial time $(2+\epsilon)$-approximation algorithm for the host evacuation problem. Section 4 introduces custom-designed scheduling heuristics, specific to the host evacuation scenario. Section 5 empirically evaluates the results and the performance of the custom heuristics and compares them to a commercial "solver". In Section 6 we discuss various future research directions.

2. RELATED WORK

Job scheduling has been extensively researched. Since in general it is *NP*-hard to compute an optimal schedule [10], a common goal is to obtain polynomial-time approximation algorithms with provable makespan bounds. Many different scheduling problems are defined, all of which are known to be *NP*-hard [10]. All problems share a common generic goal: obtaining a schedule with the best possible makespan, using an approximation algorithm. Bounds on the ratio between the approximated and optimal makespan are of interest.

The migration scheduling problem defined in this paper is equivalent to the problem of scheduling file transfers among nodes in a network, introduced by Coffman et al [6]. Each file in a set of files needs to be transfered from a given source node to a given destination, without going through intermediate nodes. Each node has a limit on the number of concurrent file transfers to/from this node. A variant of the file transfer problem, in which in addition to node concurrency limits there are also communication concurrency limits, is presented by Nakano and Nishizeki [20].

A related problem is the classic multiprocessor scheduling problem, which concerns the scheduling of a set of jobs on a set of processors. Several variants of the problem have been investigated, among others by Graham [11], who considers partial order among jobs, and by Harche and Seshadri [12], who impose processor capacity limits.

Among a variety of known approximation algorithms, or scheduling heuristics, two are most widely used: *LS* (List Scheduling) and *LPT* (Longest Processing Time first), introduced by Graham [11]. Both heuristics are greedy, and schedule jobs according to an ordered list of all jobs. In *LPT*, the list is ordered by non-increasing job running times. In *LS*, the list order is arbitrary, which allows any greedy schedule to be generated.

Approximation bounds on the makespan for file transfer, using *LS* and *LPT*, are shown in [6]. In this paper we examine both heuristics in the context of host evacuation, a private case of migration scheduling (and file transfer), obtaining improved approximation bounds, as outlined in the summary of our contributions in Section 1.2.

Scheduling heuristics need to know the length (duration) of each live migration. A priori, only an estimate can be provided. In a recent study, Akoush et al [1] have analyzed the characteristics affecting migration performance, and presented simulation models that are shown to obtain highly accurate predictions of migration length.

The performance of scheduling heuristics is commonly evaluated as a function of the number of processors (or hosts) and any capacity limits imposed by the problem definition, while the jobs are considered to be arbitrary or unknown. There are, however, results that depend also on characteristics of the jobs. Kao and Elsayed [17] examine the performance of *LPT* as a function of the number of jobs, under constraints on the relative lengths of the jobs. In stochastic analysis of scheduling approximations, the jobs are set of independent random variables, as exemplified in the work of Coffman et al [7], and Kao and Elsayed [17]. In this paper, we show a generic upper bound on greedy scheduling algorithms for host evacuation, which takes into account the volume (sum of lengths) of migrations and the maximum migration length.

Polynomial time approximation schemes (PTAS) for multiprocessor scheduling are explored by Hochbaum and Shmoys [15] and by Alon et al [2]. PTAS is a family of approximation algorithms, in which for every ϵ there exists a polynomial-time algorithm whose solution is at most $1 + \epsilon$ times larger than the optimal, also called $(1+\epsilon)$-approximation. In this paper, we leverage the PTAS of multiprocessor scheduling in the construction of a family of $(2+\epsilon)$-approximation algorithms for host evacuation.

A different approximation algorithm, called *MF* (Multifit), is studied by Coffman et al [5]. It combines bin packing methods and binary search over the bin capacity. Further analysis of *MF* is given by Lee and Massey [18]. In another work, Lee and Massey [19] combine *LPT* and *MF*, obtaining an algorithm with improved bounds.

A variation of the file transfer problem, called data migration, considers the migration of a set of data objects among storage devices in a fully connected SAN (Storage Area Network). Anderson et al [3] show approximation algorithms using graph edge coloring methods, for versions of the problem with bypass nodes (interim storage devices). Gandhi et al [9] use linear programming methods in approximating the minimal average completion time over all nodes.

Call scheduling is another related problem domain in which approximation algorithms are applicable. A set of requests for calls with given bandwidth and duration requirements need to be scheduled in a communication network where links have predefined bandwidth capacity. The entire path between the two endpoints of the call needs to have sufficient free bandwidth at the same time. Erlebach and Jansen [8] present variants of the *LS* algorithm for call scheduling in star and tree topologies.

As background on live migration of virtual machines, significant research exists on the design, implementation and performance of live migration mechanisms. The work of Clark et al [4] defines many of the common principles of live migration, including the basic stages of the live migration process. Additional coverage includes, among others, Zhao and Figueiredo [22], Harney et al [13], and Hines and Gopalan [14]. These papers present different approaches to live migration design and variants of the mechanisms, and empirically evaluate various aspects of performance using experimental implementations. A survey of migration techniques is given by Venkatesha et al [21].

3. APPROXIMATION ALGORITHMS

The optimal makespan for host evacuation can be approximated using different heuristics. In this section we examine the properties of two well known greedy approximation algorithms.

A scheduling algorithm is *greedy* if it will schedule another job as soon as the constraints of the problem allow it. The resulting schedule is called *greedy schedule*. In the context

of host evacuation, jobs are migrations, and constraints are the concurrency limits of the source and destination hosts of any unscheduled migrations.

The two greedy scheduling algorithms we examine use the same approach: given an ordered list of all migrations, select for scheduling the first feasible migration on the list. The ordered list of all migrations is considered the heuristic of the algorithm. The first algorithm, called *List Scheduler (LS)* [6], uses an arbitrarily ordered list. The resulting schedule is called *list schedule*. The second algorithm, called *Longest Processing Time First (LPT)* [11], uses a list sorted in non-increasing order of migration lengths. The resulting schedule is called *LPT schedule*.

3.1 Transfer Channels

A migration schedule can be arranged in a number of *transfer channels* (or *channels*, in short), such that each migration is associated with a specific channel, and the migrations in any given channel do not overlap in time. Migrations from different channels may overlap.

The number of channels is determined by the concurrency limits of the hosts, and is not affected by the migrations. It is easy to see that the effective concurrency C, defined in Section 1.1, is a necessary as well as sufficient number of channels. In other words, given any set of migrations and any schedule, each migration can be assigned to one of C channels. Channel assignments could be handled by the scheduling algorithm.

Without loss of generality, a schedule can be assumed to proceed along a finite set of discrete *time points*, each of which is the start time or end time of some migration. Each Migration m_i starts at time point $t(m_i)$ and ends at time point $t(m_i)+l(m_i)$. The corresponding channel, source host, and destination host are said to be *engaged* in the migration at time points t such that $t(m_i) \leq t < t(m_i) + l(m_i)$.

It should be observed that by this definition, a channel or host are not engaged in a given migration at the time point where the migration ends. Thus, a channel is engaged in at most one migration at any time point in the schedule, whereas a destination host h_i is engaged in at most $c(h_i)$ migrations at any time point.

A channel is said to be *free* at time point t if it is not engaged in any migration at time t. Destination host h_i is said to be *available* at time point t if it is engaged in less than $c(h_i)$ migrations at t.

A *gap* in a given channel is defined as a maximal span of time points at which the channel is free, followed by a time point where the channel is engaged in a migration. Similar to a migration, a gap has a start time, length, and end time. By definition, the end time of a gap must be the start time of another migration in the same channel.

3.2 Optimality of List Scheduler

In [6] it was shown that for general migration topologies (where migrations can have arbitrary source and destination hosts), there exist instances whose optimal schedule cannot be achieved using the *LS* heuristic, for any list ordering. In other words, such instances are inherently non-greedy.

We now show that when restricting the topology to single host evacuation, an optimal list schedule always exists.

The following terminology is used with regard to a given schedule.

A migration is called *promotable* if it can be moved to an earlier start time on any channel, without making additional changes in the schedule, and without violating the concurrency limits of the corresponding source and destination hosts. By definition, a greedy schedule does not have any promotable migrations. The moving of a migration to an earlier start time is called *promotion*.

Let $t > 0$ be a time point in the schedule. A migration is said to be *blocked* at time t if it starts at t, and the corresponding destination host is not available for additional migrations at both t and the immediately preceding time point. It should be observed that by this definition, there exists another migration to the same destination that ends at t, thereby enabling the blocked migration to start.

Let t be a time point in the schedule. The *t-suffix* of a given channel is the sequence of migrations in that channel whose start time is t or later. If there are no such migrations, the t-suffix is *empty*.

Let $\widehat{t}$ be the end time of the last migration in a channel. The channel is said to be *$\widehat{t}$-terminated*. Clearly, the $\widehat{t}$-suffix of the channel is empty. It should be observed that by definition of gap, the span of time from $\widehat{t}$ onward is not a gap.

LEMMA 1. *If a schedule has no gaps and no promotable migrations, the schedule is greedy.*

PROOF. Suppose the schedule is not greedy and there are no gaps. To prove the claim, we need to show that there exists a promotable migration.

Since the schedule is not greedy, there exists a time point t' in the schedule at which some channel ch_a is free, and there exist one or more migrations with start time greater than t', whose corresponding destination host is available at time t'. Let m be the migration with the earliest start time among these migrations. Clearly, destination host $d(m)$ is available at any time point t, $t' \leq t < t(m)$. Since there are no gaps, it must be that the t'-suffix of channel ch_a is empty. Therefore, migration m is promotable to start time t' in channel ch_a. □

THEOREM 2. *Given a non-greedy host evacuation schedule, a greedy schedule can be generated which has smaller or equal makespan.*

PROOF. We transform the given schedule into a greedy one, using an iterative process. Each iteration first promotes any promotable migrations, then swaps suffixes between channels to render additional migrations promotable.

This continues until there are no promotable migrations and no gaps. The following algorithm generates the greedy schedule.

PROMOTE AND SWAP ALGORITHM:
Starting with the original schedule, perform Step 1 followed by Step 2 repeatedly until there are no promotable migrations and no gaps.

1. PROMOTE: Promote any promotable migrations to the earliest possible start time on any channel. This may open up new gaps and render additional migrations promotable, but the makespan will not increase. Continue promoting migrations in arbitrary order, until eventually no migration is promotable.
2. SWAP: Select a gap with the latest end time t. Let ch_a be the channel in which this gap is located. Let N_s^t be the number of migrations that start at time t on any channel, and N_e^t the number of migrations that end at t on any channel. By definition of gap, a migration starts in channel ch_a at time t, hence $N_s^t \geq 1$.

   Case A: If $N_s^t > N_e^t$, there exists a migration m in another channel ch_b, which starts at time t and is not blocked. Swap t-suffixes between channels ch_a and ch_b. Now migration m is promotable (into the gap in channel ch_a).

   CASE B: If $N_s^t \leq N_e^t$, there exists a channel ch_b that is t-terminated (by choice of t, there are no gaps beyond time t on any channel). Move the t-suffix from channel ch_a to ch_b. This effectively eliminates the gap in channel ch_a.

Consider a typical iteration of the algorithm. If there exist promotable migrations at the beginning of the iteration, at least one such migration is promoted in Step 1. Then in Step 2, either some gap is eliminated (Case B), or some migration becomes promotable (Case A), to be promoted in the following iteration. A new gap can be opened up only in Step 1, by promoting a migration. The overall number of possible promotions is finite (since the number of time points in the schedule is finite and there are no demotions), therefore the algorithm is guaranteed to terminate.

By Lemma 1, the resulting schedule is greedy. The algorithm never demotes any migration, therefore the makespan of the resulting schedule cannot be larger than that of the original schedule. □

COROLLARY 3. *Given host evacuation schedule S, there exists a list schedule with smaller or equal makespan.*

PROOF. By Theorem 2, a greedy schedule S' can be generated by applying the *Promote and Swap* algorithm on S. Define a list of all migrations ordered by non-increasing start times in the schedule S'. It is easy to see that with this list the *LS* algorithm will generate the identical schedule S'. □

COROLLARY 4. *An optimal list schedule exists for any host evacuation scenario.*

PROOF. This follows from Corollary 3 when schedule S is optimal. □

COROLLARY 5. *A greedy schedule can always be arranged in channels such that there are no gaps.*

PROOF. By definition, a greedy schedule has no promotable migrations, in any channel arrangement. Given an initial channel arrangement with gaps, apply the *Promote and Swap* algorithm to the schedule. Consider an iteration that starts with a greedy schedule, such as the first iteration. Since there are no promotable migrations, Step 1 is not applicable. Step 2 rearranges the channels, but does not change the schedule in terms of makespan or start times. Thus, Case A is not applicable, as it would have resulted in a promotable migration. Case B, on the other hand, eliminates one gap. Therefore, the next iteration will start with the identical greedy schedule, in a channel arrangement that has one less gap, and the argument can be repeated. When the algorithm terminates, there are no gaps. □

Based on Corollary 5, we will henceforth assume without loss of generality that a greedy schedule has no gaps.

3.3 Approximation Bounds for Greedy Schedulers

In this section we present a worst-case analysis of schedules generated by greedy schedulers, in terms of bounds on the ratio between the makespan of the schedule they generate and the optimal makespan. This is called the *approximation ratio*.

We use the following denotations, in addition to those defined in Section 1.1.

The sum of lengths of all migrations whose destination host is h_j, called the *volume* of h_j, is denoted $V(h_j)$. The sum of lengths of all migrations in the schedule, called *total volume*, is denoted $V(M)$. If the problem instance is implied, we denote the total volume V.

For a given schedule, $m^*(h_j)$ denotes the migration that ends last at destination host h_j. Without loss of generality, we assume there is a single such migration (otherwise pick one arbitrarily). Let $t_j^* = t(m^*(h_j))$ denote the start time of $m^*(h_j)$.

For a given schedule, m^* denotes the migration that ends last in the schedule. In other words, $t(m^*) + l(m^*) = L$. Without loss of generality, we assume there is a single such migration (otherwise pick one arbitrarily). Let $h^* = d(m^*)$, $t^* = t(m^*)$ and $l^* = l(m^*)$ denote, respectively, the destination host, start time, and length of m^*. We also use $c^* = c(h^*)$, and $V^* = V(h^*)$ to denote, respectively, the concurrency limit and the volume of migrations of h^*.

The length of the longest migration in the problem instance is denoted l_{max}. The length of the longest migration with destination host h_j is denoted l_{max}^j.

We denote by $t_S(m)$ and $e_S(m)$ the start time and the end time of migration m in schedule S.

We define a *front* of S towards destination host h_j at time t, denoted $F_S(h_j, t)$, to be the set of end times of all the migrations to h_j having start time before t and end time at or after t. Let $F^*_S(t) = F_S(h^*, t)$ denote the front towards h^*. Notice that $|F_S(h_j, t)| \leq c(h_j)$, because no two migrations whose end times are in this front can use the same channel.

Given two fronts, F and F' (not necessarily for the same schedule or time), we say that F *precedes* F', denoted $F \prec F'$, if there exists a bijective mapping from F to F', $\alpha : F \to F'$, such that $\forall t \in F$, $t \leq \alpha(t)$. Informally, this means that for each migration m whose end time is in F there exists a corresponding migration m' whose end time is in F', such that m' ends no earlier than m. By definition of bijective mapping, if $F \prec F'$, the cardinality of the two fronts must be the same.

The earliest time point in the schedule at which some channel is free is denoted $\bar{t}$.

LEMMA 6. *For any greedy schedule and any destination host h_j, if $\bar{t} < t^*_j$, then at any time point t such that $\bar{t} \leq t < t^*_j$, exactly $c(h_j)$ channels are engaged in migrations to h_j.*

PROOF. Assume there exists a time point t ($\bar{t} \leq t < t^*_j$) at which less than $c(h_j)$ channels are engaged in migrations to h_j. Moreover, assume that t is the latest such time point. We will show that the schedule has a promotable migration, contradicting the greedy nature of the schedule.

Following Corollary 5, we can assume without loss of generality that there are no gaps in the schedule. Hence, there exists a channel ch_a that is $\bar{t}$-terminated. In other words, the channel is free at any time from $\bar{t}$ and onward.

By choice of t, some migration to host h_j (possibly $m^*(h_j)$) starts at the time point immediately following t. This migration is promotable to time t in channel ch_a, since at time t host h_j is engaged in less than $c(h_j)$ migrations, and the channel is free. □

We define γ to be an upper bound on the ratio between the length of the migration that ends last, and the optimal makespan.

$$\frac{l^*}{OPT} \leq \gamma$$

As will be seen in Theorem 8, the approximation bound of a greedy scheduling algorithm can be formulated as a function of γ. Clearly, $0 < \gamma \leq 1$. Scheduling algorithms may have inherent bounds on γ, which are dependent only on the characteristics of the algorithm. Let γ_{ALG} be the inherent upper bound on γ for a given scheduling algorithm ALG. For the LS algorithm $\gamma_{LS} = 1$, and this is tight. An example of a list schedule with $l^* = OPT$ is shown in Figure 2, therefore an upper bound which is smaller than 1 cannot exist. If no inherent bound for ALG is known, we always use $\gamma_{ALG} = 1$.

When the volume of migrations and the longest migration of the problem instance are known, the value of γ can be refined. Since $l^* \leq l_{max}$ and $OPT \geq V/C$, it is easy to see that:

$$\gamma \leq \frac{l_{max} C}{V}$$

Similarly, for any destination host h_j:

$$\gamma \leq \frac{l^j_{max} c(h_j)}{V(h_j)}$$

Hence, for greedy scheduling algorithm ALG and a given host evacuation problem instance,

$$\gamma = \min\left(\gamma_{ALG}, \frac{l_{max} C}{V}, \max_{h_j}\left(\frac{l^j_{max} c(h_j)}{V(h_j)}\right)\right) \tag{1}$$

LEMMA 7. *If $\bar{t} \geq t^*$, then the upper bound on the makespan of any greedy scheduling algorithm for host evacuation is:*

$$L \leq \left(1 + \gamma - \frac{\gamma}{C}\right) OPT \tag{2}$$

PROOF. The proof is similar to the upper bound for multiprocessor scheduling [11], using $\frac{l^*}{OPT} \leq \gamma$ instead of $\frac{l^*}{OPT} \leq 1$.

Before time point $t^* = L - l^*$, all channels are engaged in migrations. Hence,

$$\begin{aligned} &(L - l^*)C + l^* \leq V \leq C \cdot OPT \\ &L \cdot C \leq l^*(C-1) + C \cdot OPT \\ &L \leq \left(\frac{l^*}{OPT}(1 - \frac{1}{C}) + 1\right) OPT \end{aligned}$$

Substituting $\frac{l^*}{OPT}$ with γ, we obtain the specified bound. □

THEOREM 8. *The upper bound on the makespan of any greedy scheduling algorithm for host evacuation is:*

$$L \leq \left(2 + \gamma - 2\sqrt{\frac{\gamma}{C}}\right) OPT \tag{3}$$

PROOF. It is easy to see that the bound of Lemma 7 (equation 2) is lower than the bound we prove here (equation 3). Therefore we need to consider only the case $\bar{t} < t^*$.

The proof is illustrated by figure 1.

Let $\Delta t = t^* - \bar{t}$, and let $V' = \Delta t \cdot c^* + l^*$. Then:

$$\Delta t = \frac{V' - l^*}{c^*}$$

It follows from Lemma 6 that $V' + \bar{t} \cdot C \leq V$, and hence:

$$\bar{t} \leq \frac{V - V'}{C}$$

Therefore:

$$\begin{aligned} L &= \bar{t} + \Delta t + l^* \\ &\leq \frac{V - V'}{C} + \frac{V' - l^*}{c^*} + l^* \\ &= \frac{V}{C} + \frac{V'}{c^*}\left(1 - \frac{c^*}{C}\right) + l^*\left(1 - \frac{1}{c^*}\right) \end{aligned} \tag{4}$$

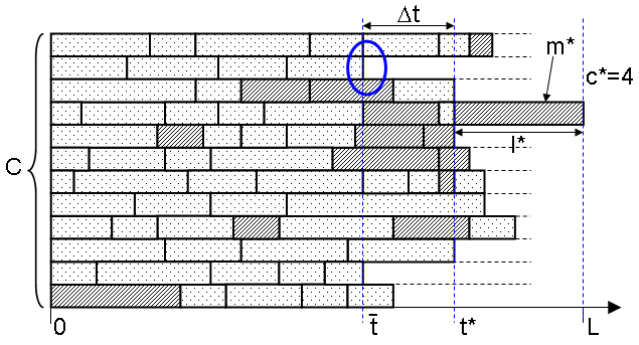


Figure 1: Illustration for Theorem 8.

We have the following upper bounds on V, V' and l^*:

$$V \leq C \cdot OPT$$
$$V' \leq V^* \leq c^* \cdot OPT$$
$$l^* \leq \gamma \cdot OPT$$

Substituting these upper bounds in equation 4, we get:

$$L \leq \frac{C \cdot OPT}{C} + \frac{c^* \cdot OPT}{c^*}\left(1 - \frac{c^*}{C}\right) + \gamma OPT\left(1 - \frac{1}{c^*}\right)$$
$$= \underbrace{\left(2 + \gamma - \frac{c^*}{C} - \frac{\gamma}{c^*}\right)}_{R} OPT \quad (5)$$

Let R denote the approximation ratio L/OPT. We can find the maximum of R for all possible values of c^*, using the derivative.

$$\frac{dR}{dc^*} = -\frac{1}{C} + \frac{\gamma}{c^{*2}}$$

Solving for derivative value 0, we get:

$$c^* = \sqrt{\gamma \cdot C}$$

Substituting for c^* in equation 5, we obtain the maximum approximation ratio:

$$R = 2 + \gamma - 2\sqrt{\frac{\gamma}{C}} \quad (6)$$

□

We will now show approximation bounds for the *LS* and *LPT* algorithms.

COROLLARY 9. *The upper bound on the makespan of the* LS *algorithm for host evacuation is*

$$L \leq \left(3 - \frac{2}{\sqrt{C}}\right) OPT \quad (7)$$

PROOF. Substitute $\gamma = 1$ in equation 6. □

THEOREM 10. *The approximation bound for* LS *given by equation 7 is* tight.

PROOF. We will show how to construct list schedules that achieve the approximation ratio specified in equation 7. An example can be seen in Figure 2.

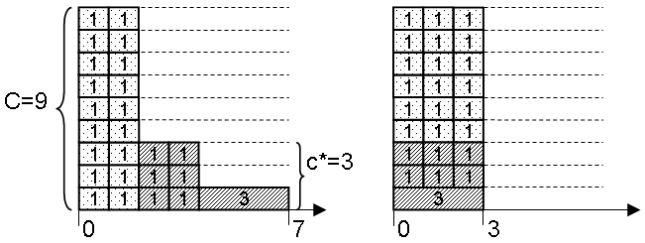


Figure 2: Example showing that the upper bound for *LS* is tight.

There are two destination hosts, the first has concurrency limit 9, and the second has concurrency limit 3. The source host has concurrency limit 9. Thus, the number of channels is 9. The list starts with 18 unit-length migrations to the first destination, followed by 6 unit-length migrations to the second destination, and finally a migration with length 3 to the second destination. The list schedule is shown on the left, and the optimal schedule on the right. Dotted rectangles are migrations to the first destination, and shaded rectangles are migrations to the second destination.

This example can be generalized, given any integer $k \geq 2$ and two or more destination hosts. One of the destination hosts, call it the last destination, has concurrency limit k, whereas all other hosts, including the source, have concurrency limit k^2. Thus, the number of channels is $C = k^2$. The list starts with $(k-1)k^2$ unit-length migrations to arbitrary destinations except the last destination (scheduling $k-1$ migrations on each of k^2 channels), followed by $(k-1)k$ unit-length migrations to the last destination ($k-1$ migrations on each of k channels), and finally a migration of length k to the last destination. By construction, the makespan of the list schedule is $3k-2$, whereas the makespan of the optimal schedule is k. This yields the approximation ratio $3 - \frac{2}{\sqrt{C}}$. □

LEMMA 11. *Given an* LPT *schedule, if* $l^* > \frac{1}{3}OPT$ *then* $L \leq 2 \cdot OPT$

PROOF. If $\bar{t} \geq t^*$, we have $L \leq \bar{t} + l^* \leq 2 \cdot OPT$ (regardless of any bound on l^*). It remains to consider the more difficult case $\bar{t} < t^*$. The proof is illustrated by Figure 3.

Let S denote the LPT schedule at hand, and let M be the set of migrations of S. Let m^*, h^*, t^*, l^* and $\bar{t}$, as defined earlier in this section, correspond specifically to the given schedule S.

Given schedule S, we now construct a subset of M and two schedules for this subset. Let $\overline{M}$ be the subset of M comprising all the migrations m having the following properties: $d(m) = h^*$, $l(m) \geq l^*_S$, and $e_S(m) > \bar{t}$. Obviously, any schedule for $\overline{M}$ uses c^* channels. Let $\overline{S}$ be an LPT schedule for $\overline{M}$ and let S' be a schedule constructed from $\overline{S}$ by shifting the start time of every migrations in $\overline{S}$ by $\bar{t}$. Thus,

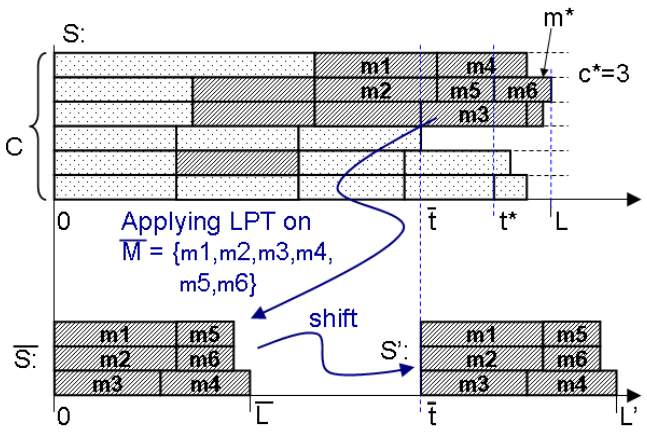


Figure 3: Illustration for Lemma 11.

S' has a gap of length $\bar{t}$ at time 0 on each channel. Finally, let $\overline{L}$ and L' be, respectively, the makespan of $\overline{S}$ and S'. By construction, $L' = \overline{L} + \bar{t}$.

Due to Lemma 6, it is easy to see that for any time point t such that $\bar{t} < t \leq t^*$, the cardinality of $F^*_S(t)$ is c^*. Also, by construction, the scheduling order among migrations in $\overline{M}$ is the same in schedules S and S' (ignoring without loss of generality any permutations among migrations having the same length). We now prove by induction on the scheduling order, that for any migration $m \in \overline{M}$ with $t_S(m) > \bar{t}$, we have the precedence $F^*_S(t_S(m)) \prec F^*_{S'}(t_{S'}(m))$. It should be observed that the migration ending last in S' is not necessarily m^*, as can be seen in Figure 3.

The induction base is the earliest migration in the scheduling order that starts after $\bar{t}$. In this case the fronts consist of the end times of the first c^* migrations (both in S and in S'). By construction of S', the first c^* migrations start at time $\bar{t}$, while in S they start at times less than or equal to $\bar{t}$. Thus, the end time of each of the first c^* migration in S is less than or equal to the corresponding end time in S', and the precedence holds.

Now, assume the precedence holds for all migrations having start times greater than $\bar{t}$ in the scheduling order of $\overline{M}$, through migration m_i. Then it must hold also when the next migration is scheduled, since the two fronts are adjusted by increasing the respective lowest end times by the same amount $l(m_i)$. This holds even if there are multiple migrations with the same start time $t_S(m_i)$, since a different member in the front will be adjusted for each of these migrations. Hence, the precedence holds for all migrations starting after $\bar{t}$, and in particular for m^*, meaning that m^* ends in S' no earlier than in S.

By construction, $e_{S'}(m^*) = e_{\overline{S}}(m^*) + \bar{t}$. Also, it is obvious that $e_{\overline{S}}(m^*) \leq \overline{L}$, and $\bar{t} \leq OPT$. Therefore,

$$L = e_S(m^*) \leq e_{S'}(m^*) = e_{\overline{S}}(m^*) + \bar{t} \leq \overline{L} + OPT \qquad (8)$$

It remains to evaluate an upper bound on $\overline{L}$. By construction, m^* is the shortest migration in $\overline{M}$. Also, it is obvious that $OPT \geq \overline{OPT}$. Thus, for any migration $m \in \overline{M}$,

$$l(m) \geq l^* > \frac{1}{3}OPT \geq \frac{1}{3}\overline{OPT}$$

It should be observed that $\overline{M}$ is equivalent to a classical multiprocessor scheduling problem with c^* processors (it contains only migrations to the single host h^*). It has been shown in [11] that for multiprocessor scheduling, if all the jobs are longer than one-third of the optimum makespan, then LPT necessarily produces an optimal schedule. In our case, this means that $\overline{L} = \overline{OPT}$. By substituting into equation 8, we obtain the claimed bound. □

THEOREM 12. *The upper bound on the makespan of the* LPT *algorithm for host evacuation is:*

$$L \leq \left(\frac{7}{3} - \frac{2}{\sqrt{3C}}\right) OPT \qquad (9)$$

PROOF. For $C = 1$, obviously any greedy schedule is optimal. For $C \geq 2$, if $l^* > \frac{1}{3}OPT$, then by Lemma 11, $L \leq 2 \cdot OPT$, which is strictly less conservative than the bound we want to prove. Otherwise, by substituting $\gamma = \frac{1}{3}$ in equation 6 we obtain the claimed bound. □

It is an open question whether the upper bound on LPT for host evacuation in equation 9 is tight.

An interesting consequence of equation 1 is that for scenarios with large volumes of migrations, the corresponding γ would be small. This yields better approximation bounds. For example, if $l_{max} = 10$, $C = 4$ and $V = 160$ (e.g., a total of 40 migrations of average length 4), the resulting γ would be 0.25. Substituting γ in equation 6, the corresponding upper bound would be 1.75, which is considerably better than 2, derived from the LS bound in equation 7, and slightly better than 1.7559, derived from the LPT bound in equation 9.

3.4 (2+ϵ)-Approximation Algorithm

Here we show how to construct a polynomial time (2+ϵ)-approximation algorithm for the host evacuation problem, using a Polynomial Time Approximation Scheme (PTAS) of the classical multiprocessor scheduling problem [15].

THEOREM 13. *Given a polynomial time $B(p)$-approximation algorithm for the multiprocessor scheduling problem with p processors, there exists a polynomial-time $(1 + B(C))$-approximation algorithm for the host evacuation problem with effective concurrency C.*

PROOF. Let ALG denote the algorithm for multiprocessor scheduling. We construct the following algorithm for host evacuation.

1. Schedule migrations using any greedy algorithm, e.g., LS with an arbitrary migrations ordering. Let S be the resulting schedule.

2. Let $\bar{t}$ be the earliest point of time in schedule S at which some channel is free. Let $\overline{H}$ be the set of destination hosts with migrations having start time at least $\bar{t}$. For each $h \in \overline{H}$, let $\overline{M}_h$ be the set of migrations with destination host h and end time greater than $\bar{t}$, and let $\overline{M} = \cup_{h \in \overline{H}} \overline{M}_h$.

3. For each $h \in \overline{H}$, construct a schedule S_h for the set of migrations $\overline{M}_h$ with $c(h)$ channels using algorithm ALG (following the straightforward conversion between the two problems).

4. For each $h \in \overline{H}$, construct a schedule S'_h from S_h by shifting the start time of each of the migrations in S_h by $\bar{t}$.

5. Let $S' = \cup_{h \in \overline{H}} S'_h$, and let $\widehat{S}$ be the sub-schedule of S for the set of migrations $\widehat{M} = M \setminus \overline{M}$. Output the schedule $S_0 = \widehat{S} \cup S'$.

Let $\overline{C} = \sum_{h \in \overline{H}} c(h)$ be the sum of concurrency limits of the set of hosts $\overline{H}$, and let $\widehat{C}$ be the number of channels that are engaged in migrations of $\widehat{M}$ at time $\bar{t}$. By Lemma 6, $\widehat{C} + \overline{C} \leq C$.

We first consider schedule $\widehat{S}$. Since no migration starts after time $\bar{t}$ in schedule $\widehat{S}$, and $\bar{t} \leq OPT$, it follows that the end time of the last migration in this schedule is at most $2 \cdot OPT$. Thus, the makespan of schedule $\widehat{S}$ is at most $2 \cdot OPT$.

We now consider schedule S'. For each host $h \in \overline{H}$, the algorithm solves a multiprocessor scheduling problem with the set of migrations $\overline{M}_h$ and $c(h)$ channels, using a $B(C)$-approximation algorithm. Thus, the makespan of each of the schedules S_h is at most $B(C) \cdot OPT$. Since each of the sub-schedules S'_h starts at time $\bar{t}$ and $\bar{t} \leq OPT$, it follows that the makespan of schedule S' is at most $(1+B(C))OPT$.

Finally, we consider schedule S_0. Since schedule $\widehat{S}$ is a sub-schedule of S, schedule S_0 complies with all the concurrency limits at any time $t \leq \bar{t}$. Since each of the schedules S' and $\widehat{S}$ complies with all concurrency limits at any time $t \geq \bar{t}$, and having $\widehat{C} + \overline{C} \leq C$, schedule S_0 complies with all the concurrency limits at any time $t \geq \bar{t}$. Thus, schedule S_0 complies with all the concurrency limits at all times. The makespan of schedules S' and $\widehat{S}$ is at most $(1+B(C))OPT$. Therefore, the makespan of schedule S_0 is at most $(1+B(C))OPT$. □

By using the PTAS for multiprocessor scheduling problem given in [15], we obtain the following corollary.

COROLLARY 14. *For any $\epsilon > 0$, there exists a polynomial-time $(2+\epsilon)$-approximation algorithm for the host evacuation scheduling problem.*

4. CUSTOM HEURISTICS

The LS and LPT heuristics were originally designed for the multiprocessor scheduling problem. In fact, when applying LPT in the context of host evacuation, the strict LPT order of all migrations is not an inherent property of the resulting schedule. On the other hand, LPT order is always maintained among the migrations of each destination separately.

With this observation in mind, we introduce a family of custom scheduling heuristics for host evacuation. Although the approximation bound of the custom heuristics is identical to LPT (see Theorem 15), in Section 5 we demonstrate empirically that the custom heuristics achieve considerably better results than LPT.

CUSTOM SCHEDULING SCHEME:
Perform the following steps repeatedly until all migrations have been scheduled.

1. For each destination host that is available (i.e., it is possible to schedule another migration to that destination without violating its concurrency limit), designate as *candidate* for scheduling the longest unscheduled migration to that host.

2. Select for scheduling one of the candidate migrations, according to a given evaluation function (see below).

3. Advance the schedule to the earliest time point where some channel is free.

It should be observed that in this scheduling scheme, migrations are not scheduled according to a sorted list of all migrations. However, the migrations of each destination are scheduled in non-increasing order of lengths.

Let us define the *residual volume* of a destination host at a given point in time, to be the sum of all unscheduled migrations to that destination. The *normalized residual capacity* is a quantity obtained by dividing the residual capacity by the corresponding concurrency limit.

We now present two basic evaluation functions, and a variant of each. The rationale behind all functions is to postpone as much as possible the point in time at which channels become underutilized. In other words, a point where only migrations to a few destination hosts remain unscheduled, such that the sum of concurrency limits of the remaining destinations gets below the the number of channels. Informally, we would like to keep the residual volumes of all destinations as close together as possible, in an attempt to defer $\bar{t}$.

Max: Select the candidate migration of the destination host with the largest residual volume.

Max-norm: Similar to **max**, except that normalized residual volumes are used instead of residual volumes.

Balanced: Select the candidate migration such that after it is scheduled, the difference between the largest and smallest residual volumes of all destinations is minimized.

Balanced-norm: Similar to **balanced**, except that normalized residual volumes are used instead of residual volumes.

THEOREM 15. *The approximation bound of the custom heuristics for host evacuation (max, max-norm, balanced, balanced-norm) is:*

$$\frac{7}{3} - \frac{2}{\sqrt{3C}}$$

PROOF. The arguments used in the proof of Lemma 11 and Theorem 12 for *LPT* are valid also for the custom heuristics, for the following reasons. Due to Lemma 6, starting with $\bar{t}$, the behavior of the custom schedule is identical to the *LPT* schedule. Moreover, the proof for *LPT* does not assume anything regarding the ordering of migrations prior to $\bar{t}$ (except for the greedy nature of the algorithm). □

It is an open question whether this approximation bound for the custom heuristics is tight.

From an algorithmic perspective, the four custom heuristics and *LPT* can all be combined into a single *hybrid* approximation algorithm, which applies each of the heuristics, then returns the schedule that achieves the smallest makespan.

5. EMPIRICAL RESULTS

In this section we examine how the scheduling heuristics perform in practice. Generating samples of host evacuation instances and applying the heuristics, we show that the custom heuristics produce optimal or close to optimal schedules in a vast majority of the attempted samples.

We apply the heuristics to samples with a small number of migrations (20–40 VMs on the evacuated host), then compare the outcome to the optimal spolution. We are able to ascertain the optimal makespan by leveraging a commercial constraint programming solver. The results support our hypothesis, that although the theoretic approximation bound of our *hybrid* heuristic is the same as *LPT* (see Section 4), our heuristic performs much better than *LPT* in practice.

We also demonstrate that the custom heuristics scale better than a generic solver. We apply the heuristics and the solver to large samples of 100–1000 migrations, while limiting the solver either to a threshold on the running time or to a threshold on the makespan. We then compare the results in favor of our heuristics.

All the scheduling algorithms we have used are implemented in Java, and all measurements were done on a standard workstation computer (Intel Core 2 Duo 2.33 GHz, 2GB of RAM).

5.1 Simulation Framework

In order to assess the various scheduling approaches, we have built an extensive simulation framework that simulates a variety of host evacuation setups, having different combinations of the following parameters:

- Number of destination hosts.
- Concurrency limit of each host.
- Number of VMs on the source host (being evacuated).
- Assignment of evacuated VMs to destination hosts.
- Sizes of the VMs (determining the migration length).

For each parameter, we have considered four statistical distributions: fixed, uniform, Gaussian and exponential. Moreover, we performed several iterations applying the same set of distributions. Overall, the analysis was done using data sets of 4050 samples each.

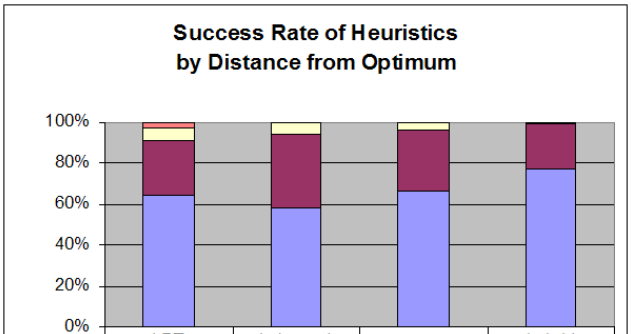
5.2 Using a Generic Solver

Aside from using common or custom heuristics, an alternative approach to solving scheduling problems is to deploy generic constraint programming software (a "solver"). In order to assess this approach, we have implemented a scheduler for host evacuation deploying the IBM ILOG CPLEX CP Optimizer [16].

The solver can be invoked with an upper threshold on the running time, or a lower threshold on the makespan. The solver then stops either when it finds the optimal solution (and is able to confirm its optimality), or when it crosses the threshold, in which case it returns the best solution found so far.

5.3 Comparison of Schedulers

For small samples of 20–40 migrations, it takes the solver few seconds to find the optimal solution. We use these results to assess the approximation quality of our custom heuristics. The comparison among the different heuristics is summarized in the table in Figure 4.



	LPT	balanced	max	hybrid
more	2.9%	0.1%	0.0%	0.0%
5-10%	5.4%	5.4%	3.5%	0.6%
0-5%	25.2%	35.4%	29.4%	21.7%
optimal	60.2%	57.5%	66.6%	77.7%

Figure 4: Comparison of heuristics by distance from optimum.

It can be seen that the *hybrid* heuristic finds the optimal solution in 77.7% of the samples, while in another 21.7% of the samples it finds a result which is less than 5% away from optimal. This leaves just 0.6% of the samples in which this heuristic produces a makespan that is away from optimal by more than 5%. Also, it can be seen that on average, the *max* heuristic produces the best results among the individual heuristics. It finds the optimal solution in 66.6%

of the samples, and achieves a makespan at most 5% from optimal in another 29.4%, compared to 57.5% and 35.4% for *balanced*, and 60.2% and 25.2% for *LPT*).

We have compared the average approximation quality of the custom heuristics under different statistical distributions of some of the sample parameters. The results are shown in Figure 5, which considers distribution of migration lengths, and in Figure 6, which considers distribution of concurrency limits of destination hosts. It can be seen that in both cases, the heuristics are most successful in finding the optimal schedule when the statistical distribution is exponential, and least successful when the distribution is normal. Moreover, it can be seen that the trend that *max* provides better approximation than other individual heuristics (but worse than their combination) holds also for data sets restricted to specific distributions. These results can be used to assess the expected approximation quality of the heuristics under specific conditions, such as when the statistical distribution of the problem parameters is known.

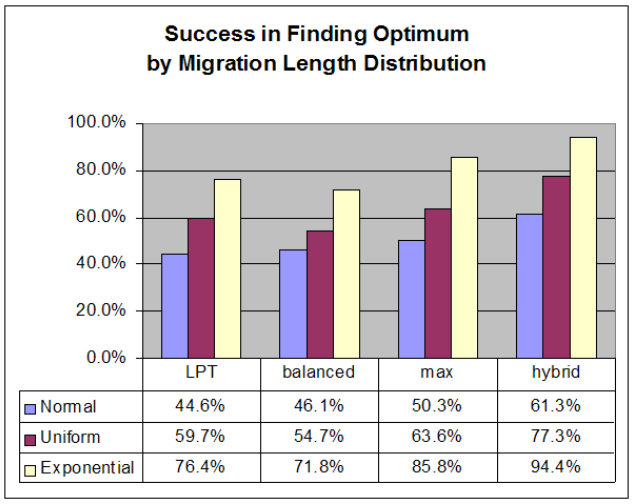


Figure 5: Comparison of heuristics by migration length distribution.

For large samples of 100–1000 migrations, we have utilized the solver in two ways: finding the optimal solution, and approximating the optimal solution by finding at least the same solution as our heuristic. We accomplished the latter by invoking the heuristic and recording the achieved makespan, then invoking the solver while aiming at finding a schedule with the same makespan. The comparison results are summarized in Figure 7. It can be seen that the custom heuristic takes significantly less time than the solver approximation, which in turn takes less time than finding the optimal solution.

To summarize, our experiments show that the *hybrid* heuristic algorithm achieves very good results for host evacuation, both in terms of accuracy, and in terms of scalability. Moreover, it significantly outperforms the *LPT* heuristic. Finally, its simplicity is an important factor when comparing with heavier tools like generic solvers.

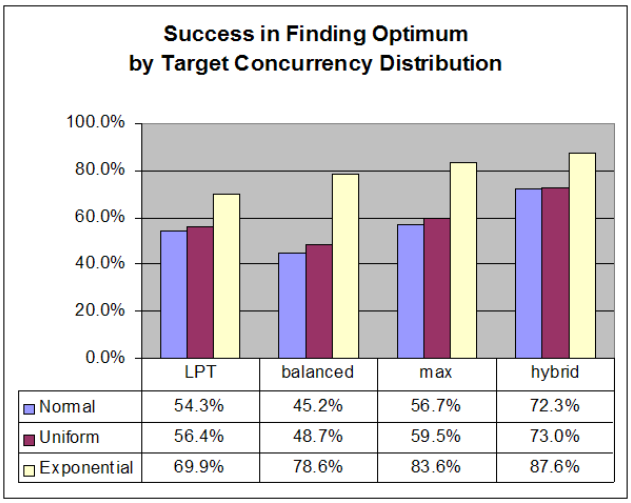


Figure 6: Comparison of heuristics by concurrency limit distribution.

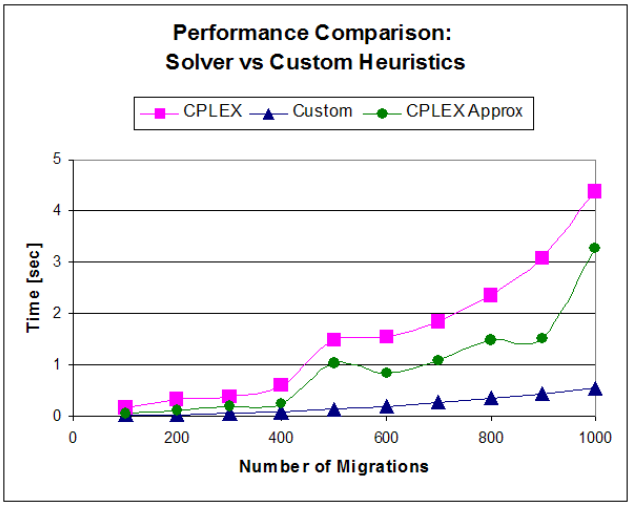


Figure 7: Performance of custom heuristic vs. solver for large samples.

6. FUTURE RESEARCH DIRECTIONS

This paper is an initial step in exploring scheduling and management aspects of VM placement changes. There are opportunities to continue this research in multiple directions.

6.1 Modeling of Resource Constraints

The migration scheduling model described in Section 1.1 can be enhanced to capture resource constraints more accurately. Instead of defining a fixed concurrency limit per each host, the following more flexible schemes could be used.

Elastic bandwidth utilization: *Utilized* network bandwidth is defined as a function of *allocated* network bandwidth, with minimum and maximum thresholds. A typical function can be: zero below minimum, linear between minimum and maximum, or flat above maximum. Migration speed is proportional to utilized bandwidth. The model used

in this paper can be considered a special case, where minimum equals maximum.

Dedicated processor and network resources: Certain amount of CPU and network bandwidth are dedicated to migration work. Multiple migrations compete for CPU cycles(on the the same host) and network bandwidth (depending on network capacity and topology).

Proportional processor and network resources: Each migration has an amount of state to be transferred which is proportional to the size of the VM. Assuming at least a minimal amount of host CPU and network bandwidth are dedicated to migration, a fixed ration of CPU and bandwidth are allocated per 1MB of state transfer.

6.2 Generalized and Alternative Models

The migration scenarios considered in this paper have limited versatility, and the scheduling model is therefore relatively straightforward. The model can be generalized to accommodate a larger scope of realistic situations.

Precedence among migrations: The current model does not accommodate precedence constraints among migrations, other than those derived from concurrency limits. This might not be realistic in environments with large number of hosts and VMs, where free resources are too fragmented. This situation imposes a constraint on the sequencing of migrations. Moreover, routing of migrated VMs via intermediate hosts may become necessary, in order to match the migrated VMs to free resources. Precedence constraints may be also be needed due to functional aspects of the applications running on the VMs. To capture such constraints, the model can be augmented with a partial order among migrations. It should be observed that there could also be requirements to interleave migrations with other placement operations, such as deployment of new VMs.

Scheduling without predefined placement: In our model, the destination host of each migrated VM is predefined. In some situations, such as emergency host evacuation due to predicted hardware failure, it is beneficial to consider a different placement scheme, where the destination of each migrated VM is derived from a schedule with minimal makespan, observing the any concurrency and capacity constraints. After all the VMs are safely evacuated from the failing host, another round of migrations can be applied, to shuffle VMs around and achieve a more optimal placement (for example, in terms of load-balancing).

Alternative optimization goals: The optimization goal most commonly used, including in this paper, is to minimize the makespan. Alternative goals can be considered, such as:

- Minimize the average completion time of all migrations.
- By assigning a weight to each migration and setting an upper bound on the overall migration time, maximize the total weight of all migrated VMs. This is relevant in situations where not all the migrations can be completed, due to time constraints (for example, outage of the evacuated host is imminent within predicted time).

Offline migration: Our migration model assumes that all VMs are online prior to evacuation, and stay online during and after migration. Hence live migration techniques are required, and concurrency constraints are imposed. In scenarios which are either highly time-constrained or resource-constrained, such as emergency host evacuation, a modified approach can be used. Only a select subset of the VMs are migrated live. The remaining VMs are either shut down or taken into hibernation at the original host (i.e., state information that needs to be migrated is moved offline). This is followed by offline migration of the VMs to their destination, where the VMs are restarted. The decision which VMs to migrate live can be made with the help of weights, as discussed above. The goal is to optimize the tradeoff between increase in overall migration time (or resource utilization) and the impact of taking some VMs offline.

6.3 Experimentation on a Real System

The empirical work described in this paper was done using a specially designed simulation framework. The advantage of this approach is that it enables experimentation with very large problem sizes. However, the accuracy of the model and performance of the algorithms should be put to test in a real system environment with actual migrations.

7. CONCLUSIONS

This paper presented the problem of scheduling live migrations of virtual machines during host evacuation. We defined a model for the problem, and analyzed the approximation bounds of two common scheduling heuristics. Although host evacuation is a private case of the more general problem of file transfer scheduling (and equivalently migration scheduling), which has been previously investigated, we showed improved approximation bounds for the private case. We also constructed a $(2+\epsilon)$-approximation scheme, and introduced several custom heuristics for host evacuation, which were shown to outperform the common heuristics in practice. This research can be extended in several directions, exploring different aspects of scheduling and management of VM placement changes.

8. REFERENCES

[1] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper. Predicting the performance of virtual machine migration. In *MASCOTS 2010: 18th annual IEEE/ACM international symposium on modeling, analysis and simulation of computers and telecommunication systems*, pages 37–46. Miami Beach, FL, Aug. 2010.

[2] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation algorithms for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, June 1998.

[3] E. Anderson, J. Hall, J. Hartline, M. Hobbesm, A. Karlin, J. Saia, R. Swaminathan, and J. Wilkes. Algorithms for data migration. *Algorithmica*, 57(2):349–380, June 2010.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *USENIX NSDI'05: 2nd symposium on networked systems design &*

implementation, pages 273–286. Boston, MA, May 2005.

[5] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, Feb. 1978.

[6] E. G. Coffman Jr., M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers. *SIAM Journal on Computing*, 14(3):744–780, Aug. 1985.

[7] E. G. Coffman Jr., G. S. Lueker, and H. G. Rinnooy Kan. Asymptotic methods in the probabilistic analysis of sequencing and packing heuristics. *Management Science*, 34(3):266–290, March 1988.

[8] T. Erlebach and K. Jansen. Off-line and on-line call-scheduling in stars and trees. In *WG'97: 23rd international workshop on graph-theoretic concepts in computer science, LNCS Vol. 1335*, pages 199–213. Berlin, Germany, June 1997.

[9] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Improved algorithms for data migration and open shop scheduling. *ACM transactions on algorithms*, 2(1):116–129, Jan. 2006.

[10] M. R. Garey and D. S. Johnson. *Computers and intractability, a guide to the theory of NP-completeness*. W. H. Freeman & Co., San Francisco, 1979.

[11] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, Mar. 1969.

[12] F. Harche and S. Seshadri. An LPT-bound for a parallel multiprocessor scheduling problem. *Journal of Mathematical Analysis and Applications*, 196(1):181–195, November 1995.

[13] E. Harney, S. Goasguen, J. Martin, M. Murphy, and M. Westall. The efficacy of live virtual machine migrations over the internet. In *VTDC'07: 2nd international workshop on virtualization technologies in distributed computing*. Reno, NV, Nov. 2007.

[14] M. R. Hines and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *VEE'09: The 2009 ACM SIGPLAN/SIGOPS international conference on virtual execution environments*, pages 51–60. Washington, DC, Mar. 2009.

[15] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, Jan. 1987.

[16] IBM Corporation. *IBM ILOG CPLEX CP optimizer, Data sheet*, Somers, NY, June 2010.

[17] T. Y. Kao and E. A. Elsayed. Performance of the LPT algorithm in multiprocessor scheduling. *Computers & Operations Research*, 17(4):365–373, 1990.

[18] C. Lee and J. D. Massey. Multiprocessor scheduling: an extension of the multifit algorithm. *Journal of Manufacturing Systems*, 7(1):25–32, 1988.

[19] C. Lee and J. D. Massey. Multiprocessor scheduling: combining LPT and multifit. *Discrete Applied Mathematics*, 20(3):233–242, 1988.

[20] S. Nakano and T. Nishizeki. Scheduling file transfers under port and channel constraints. *International Journal on Foundations of Computer Science*, 4(2):101–115, June 1993.

[21] S. Venkatesha, S. Sadhu, and S. Kintali. Survey of virtual machine migration techniques. Technical report, Dept. of Computer Science, University of California, Santa Barbara, 2009.

[22] M. Zhao and R. J. Figueiredo. Experimental study of virtual machine migration in support of reservation of cluster resources. In *VTDC'07: 2nd international workshop on virtualization technologies in distributed computing*. Reno, NV, Nov. 2007.