On Selecting the Right Optimizations for Virtual Machine Migration

Senthil Nathan, Umesh Bellur *, and Purushottam Kulkarni
Department of Computer Science and Engineering, IIT Bombay
{cendhu,umesh,puru}@cse.iitb.ac.in

Abstract
To reduce the migration time of a virtual machine and network traffic generated during migration, existing works have proposed a number of optimizations to pre-copy live migration. These optimizations are delta compression, page skip, deduplication, and data compression. The cost-benefit analysis of these optimizations may preclude the use of certain optimizations in specific scenarios. However, no study has compared the performance & cost of these optimizations, and identified the impact of application behaviour on performance gain. Hence, it is not clear for a given migration scenario and an application, what is the best optimization that one must employ?

In this paper, we present a comprehensive empirical study using a large number of workloads to provide recommendations on selection of optimizations for pre-copy live migration. The empirical study reveals that page skip is an important optimization as it reduces network traffic by 20% with negligible additional CPU cost. Data compressions yields impressive gains in reducing network traffic (37%) but at the cost of a significant increase in CPU consumption (5×). Deduplication needs to be applied with utmost care as the increase in CPU utilization might outweigh the benefits considerably. The combination of page skip and data compression works the best across workloads and results in a significant reduction in network traffic (40%).

1. Introduction
Live virtual machine (VM) migration [5] enables the movement of a VM from one physical server to another while the VM is executing. The migration process achieves this by transferring both memory pages and execution state of the VM. As VM’s execution modifies memory state, memory pages are transferred over multiple iterations. After a certain number of iterations, the migration process suspends the VM’s execution to transfer the residual pages and execution state. The performance metrics of migration are migration time (the time taken to complete the migration), downtime (the duration for which the VM is suspended), and network traffic (the amount of data transferred).

A data center employs VM migration for various management tasks including mitigating resource hotspots on an overloaded server [24], and evacuating a server for software/hardware upgradation. In each of these cases, an important requirement is to perform VM migration quickly. Further, it is necessary to reduce the network traffic generated during migration as this can cause performance degradation in other VMs [18]. Existing works [9, 18, 22, 23] have proposed various optimizations to reduce the migration time and network traffic. These optimizations are (i) delta compression—send only the modified page content instead of the entire page, (ii) deduplication—avoids the transfer of duplicate and zero pages, (iii) page skip—avoids transferring frequently dirtied pages, and (iv) data compression—compress pages by exploiting word level duplicate.

These optimizations come at the cost of increased CPU and memory utilization. The additional resources consumed by the migration process may become harmful in scenarios such as when the migration is being used to mitigate a resource hotspot. For example, employing page skip, data compression and deduplication during migration can increase the CPU utilization by 18×. Hence, all optimizations cannot be blindly applied to every migration scenario. A thorough understanding of the relation between performance and cost is necessary to decide a specific optimization that will yield more benefit than harm for a given scenario.

Further, the improvement in migration performance with each optimization is dependent on the application behaviour. Application behavior that may impact the performance gain (as identified in this paper) are page dirty characteristics — (page dirty frequency & page dirty rate), disk read/write rate, maximum writable working set size [17], and page content similarity. For example, if an application typically modifies most content of a page, employing an optimization which painstakingly sends only the modified portion of a page is overkill, and results in wastage of resources. To maximize
the performance gain and reduce resource cost, we need to select a suitable combination of optimizations with respect to a given application behaviour and migration scenario.

However, nowhere in the current literature, we have found evidence of such a study that: (a) understands how application behavior impacts the performance gain achieved by each optimization, and (b) provides a detailed cost-benefit analysis of different optimizations (which can aid us to select suitable optimizations for a given migration scenario such as hotspot mitigation). Our goal is to perform such a study. Specifically, the three major contributions of our study are:

1. A detailed empirical evaluation of four existing optimizations for pre-copy live migration that (i) quantifies the performance and cost using 294 migration instances (over 42 workloads\footnote{The migration codes and logs are available at \url{https://goo.gl/Ikycz9}}, and (ii) correlates application behaviour with the performance gain of optimizations.

2. Discovery of a new optimization component called false dirty page that helps in (i) identifying the impact of application behaviour on the performance gain, and (ii) finding the best optimization.

3. Recommendations on (i) how to combine various optimizations for reducing the impact of application behaviour, and the resource cost; (ii) which optimizations to employ for various migration scenarios.

The rest of the paper is structured as follows: Section 2 provides a background on vanilla live migration, describes existing optimizations, and motivates an empirical study. Section 3 explains our experimental methodology and expectations on the performance of existing optimizations. Section 4 empirically evaluates each optimization to understand the performance and cost tradeoff as well as relation between application behaviour and optimizations. Section 5 utilizes insights from empirical study to propose appropriate combinations of various optimizations. Section 6 lists a set of guidelines on the usage of these optimizations for various migration scenarios while section 7 concludes the paper.

2. Background

In this section, first, we provide background for vanilla pre-copy live migration technique and existing optimization techniques that improve its performance. Next, we motivate a comprehensive empirical study.

2.1 Vanilla Pre-Copy Live Migration

Pre-copy live migration \cite{5} involves two phases—iterative pre-copy and stop-and-copy.

**Iterative Pre-Copy Phase.** Since an active VM can modify content stored in memory, the migration process transfers memory pages allocated to the VM over multiple iterations. In the first iteration, all memory pages are transferred to the destination server. In later iterations, memory pages that are modified (a.k.a dirtied) during the previous iterations are transferred. In order to identify dirtied pages, all pages are located to the VM are marked as read only by the hypervisor before starting an iteration. As a result, a trap is generated when the VM tries to modify its memory content. The hypervisor, on receiving this trap, marks the corresponding pages as dirtied and provides write permission to the VM for that page (i.e., only the first write to a page is trapped during the course of an iteration). As the change in memory state is a continuous process for an active VM, the migration process employs the stop-and-copy phase.

**Stop-and-Copy Phase.** In this phase, the execution of VM is suspended to transfer all pages dirtied during the final pre-copy iteration and also the hardware state associated with the VM. The shift from iterative pre-copy phase to stop-and-copy phase occurs when certain pre-defined condition is satisfied at the end of an iteration. The migration process stops iterative pre-copy phase if the progress across two iterations is insufficient \cite{19}, i.e., the number of pages dirtied does not reduce by over 10%.

2.2 Optimizing Pre-Copy Live Migration

In this section, we explain the details of the four existing optimizations for pre-copy live migration, (i) delta compression \cite{22}, (ii) page skip \cite{18}, (iii) deduplication \cite{23}, and (iv) data compression \cite{9}. For all optimizations, pages are considered one at a time, the optimization is applied, and then the page is transferred to the destination server. There are techniques \cite{4,8,10,12} which improve migration performance by avoiding the transfer of free pages and buffer caches but it requires guest OS modifications. Hence, we have omitted them from our study.

1. **Delta Compression.** In every pre-copy iteration, instead of transferring the entire content of a dirtied page, the delta compression technique \cite{22} transfers only the modified content of a page (called page delta—termed \texttt{cmd} as part of our study). To find the exact modified portion of a page, delta compression stores the content of the page in a cache before its transmission in an iteration. In the next iteration, while transferring a dirtied page, the migration process performs a bitwise XOR operation between the cached content of that page and the current content. Performing run length

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Workload</th>
<th>Reduction in MT</th>
<th>Reduction in NT</th>
<th>additional CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta Comp.</td>
<td>LMbench</td>
<td>26%</td>
<td>—</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>SAP ERP</td>
<td>40%</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Page Skip</td>
<td>RUBiS</td>
<td>30%</td>
<td>30%</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Kernel Compile</td>
<td>48%</td>
<td>48%</td>
<td>—</td>
</tr>
<tr>
<td>Sub Page Dedup.</td>
<td>Kernel Compile</td>
<td>15%</td>
<td>15%</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>TPC-W</td>
<td>20%</td>
<td>20%</td>
<td></td>
</tr>
<tr>
<td>Data Comp.</td>
<td>MMer</td>
<td>39%</td>
<td>48%</td>
<td>30%</td>
</tr>
<tr>
<td></td>
<td>dbench</td>
<td>39%</td>
<td>80%</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Performance gain as reported in literature. MT: Migration Time, NT: Network Traffic, —: not evaluated.
encoding on XORed data results in a compressed page. Before transferring the compressed page, the cached content is replaced with the current content. On the destination server, the page content is updated by performing bitwise XOR between the received and stored content.

(2) **Page Skip.** A virtual machine can have memory pages that are modified frequently (called hot dirty pages—hdp). The page skip technique [18] avoids transferring frequently dirtied pages to improve the performance of migration. During an iteration, if pages that are scheduled for transfer are dirtied before their transmission, the page skip technique expects these pages to get dirtied again. Hence, after transferring every m pages in an iteration, the migration process retrieves the dirty bitmap to skip all dirtied pages that are yet to be transferred in the current iteration. The work in [17] establishes an analytical model for page skip.

(3) **Deduplication.** This technique [23] identifies and transfers only one copy of duplicate content to the destination server. Content similarity can be identified for either whole page (includes zero page—zp, duplicate page—dup) or fixed sub page (includes zp, sub zero page—szp, dup, sub duplicate page—sdup) by computing hash values.

In every iteration, before transferring a page, the migration process computes the SHA-1 value of the content stored in the page. Then, by employing the SuperFashHash [1] on the computed SHA-1 value, the migration process performs a look up on the hash table. On a unsuccessful lookup, the whole page is transferred and the corresponding SHA-1 value is inserted into the hash table along with its page identifier (old entry is deleted from the hash table). On a successful hash lookup, only the page identifier of the original and duplicate pages is transferred. At the destination, either the newly received content or the existing content (in case the page is a duplicate) is used. The SHA-1 is the widely used hash function [11, 13, 14] for deduplication. For sub-page deduplication, SHA-1 value needs to computed for every sub-page of a page. To reduce collision in the hash table, we use a hash table of size equal to the number of pages allocated to the VM and the SuperFashHash [1].

(4) **Data Compression.** This optimization exploits word level duplicates (wl) in the data to reduce amount of data to be transferred. The migration process employs LZO compression algorithm [20] per page during migration to identify word level duplicates and reduce the network traffic.

### 2.3 Optimization Components

Our analysis reveals that each optimization improves the performance of pre-copy migration by eliminating overhead due to several distinct components. The original work that proposed these optimizations have not analyzed these components in detail but they are important since the presence or absence of these components are driven by application behavior. We can characterize these components into two distinct sets: those due to to either page dirty characteristics or page content characteristics of the application.

Table 2 shows the list of components that each of the optimization works on to improve the performance of migration. It is evident from Table 2 that different techniques overlap with each other in terms of the components they act upon. Note that sub page deduplication technique aids in finding page delta only at a sub page size granularity. We have discovered a new component called false dirty pages (fdp) which has been overlooked in all existing studies.

**False Dirty Page.** Ideally, the dirty logging for a page should be enabled only after that page is transferred to the destination. However, both Xen and KVM enable dirty logging for all pages allocated to the VM before every iteration. This is because, removing the write permission on a page (setting read only permission to corresponding page table entry) requires flushing of TLB entries. Frequently flushing the TLB can degrade application performance, and hence, all pages are set to read only before every iteration. As a result, in an iteration, a page that is already dirtied during the same iteration can be transferred. In the next iteration, this dirtied page is again transferred as it marked in the set of dirtied pages of the previous iteration. If this page is not dirtied again between the transfer in previous iteration and the current iteration, we end up transferring unmodified content unnecessarily. We term such pages as false dirty pages.

*Existing studies failed to notice false dirty pages altogether.* However, due to inherent nature of delta compression, page skip, and data deduplication technique, the redundant transfer of false dirty pages is avoided. It is necessary to identify the improvement due to fdp explicitly to find the impact of application behavior on performance gain.

### 2.4 Why Do We Need Yet Another Empirical Study?

In existing studies [9, 18, 22, 23], the reduction in migration time (MT) and network traffic (NT) due to each optimization

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Account for</th>
<th>Parameters</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dirty characteristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>content characteristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delta Comp. [22]</td>
<td>✓ ✓</td>
<td>Cache size</td>
<td>CPU, Memory</td>
</tr>
<tr>
<td>Page Skip [18]</td>
<td>✓</td>
<td>Build Frequency</td>
<td>CPU</td>
</tr>
<tr>
<td>Full Page Dedup. [23]</td>
<td>✓ ✓</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>Sub Page Dedup. [23]</td>
<td>✓ ✓</td>
<td>Page Granularity</td>
<td>CPU, Memory</td>
</tr>
<tr>
<td>Data Comp. [9]</td>
<td>✓ ✓</td>
<td></td>
<td>—</td>
</tr>
</tbody>
</table>

Table 2. Components accounted in each optimization, parameters affecting the performance, and cost metrics. The table lists the techniques, the components they act upon, and the parameters and cost metrics affected by each technique.
is measured against the vanilla pre-copy live migration with only a limited number of workloads for a fixed migration rate (or transfer rate). Table 1 summarizes the improvement reported in existing studies. It is necessary to consider a wide range of workloads to obtain a variety of page dirty characteristics, and should use different migration rate (one of the major factor affecting the migration’s performance [18]). Further, they have missed out on studying the effects of parameters on the performance gain. Hence, the best setting for these parameters is unknown.

Though the performance gain due to each optimization is presented in existing studies, the following two metrics are not reported: (i) the improvement due to individual components (which can help us to identify the impact of application behavior), and (ii) additional resource utilization with certain optimization (refer to Table 1). As there is no comprehensive empirical evaluation and comparison of these optimizations in terms of performance improvement and cost, answers to the following questions are also unclear.

- For a given migration scenario such as CPU or memory or network resource hotspot, which is the best optimization in terms of performance & cost? In other words, can we trade-off CPU for network resource?
- Is the increase in resource cost proportional to the performance gain?
- Does the performance gain depend on application behavior?
- How to combine various optimizations as there are overlap of components between optimizations?
- Can the combination of optimizations reduce the resource cost of individual optimization?

### 3. Experimental Methodology

In this section, first, we present the setup and workloads used. Next, we present our expectations on each optimization’s performance, and steps employed for our study.

#### 3.1 Setup and Workloads

Our setup consists of three servers, each equipped with a 2.8 GHz Intel Core i5 760 CPU (4 cores), and 4 GB of memory. One server acts as a controller that issues migration commands and generates load for the application executing in the VM. The other two servers execute QEMU-KVM v1.7. All servers are connected through a 1 Gbps switch. Each server and VM is installed with Linux kernel v3.8.0-29. Though our setup is small, our results remain valid for large setups.

This is because, we assume that the network bandwidth is reserved between the source and destination server for migration process using SDN or managed hardware switches. Further, we assume that required CPU [17] is reserved for the migration process using cpulimit tool. Hence, the interference due to other VMs can be avoided in large setups.

The three parameters that impact the performance of pre-copy live migration are (i) page dirty rate, (ii) VM memory size, and (iii) migration rate [18]. To experiment with a variety of page dirty rates, we consider a wide variety of workloads (42 in total) that are commonly hosted in data centers. Further, we use different workloads to obtain a variety of application behaviors. The workload consists of (i) web and database services—HTTP file server, RUBiS [21], Mediawiki [15], OLTPBenchmark [6], (ii) multimedia and data mining benchmarks such as NU-MineBench [16], Parsec [2], and (iii) other multi-threaded benchmarks [3]. Refer to Appendix A for a detailed description of these workloads.

All workloads are hosted on VMs of memory size 1 GB except for OLTPBenchmark and NU-MineBench which are assigned 1.5 GB and 600 MB, respectively. The VMs are migrated with 7 different transfer rates (as the migration performance is directly proportional to the transfer rate) ranging from 100 Mbps to 700 Mbps in steps of 100 Mbps to generate 294 migration instances. For each instance, we measure migration time, downtime and network traffic.
3.2 Expectations on Migration Performance

From Table 2, we have seen that the delta compression & page skip techniques account only for page dirty characteristics. Whereas, the data compression technique accounts only for page content characteristics. On the other hand, the deduplication technique accounts for both page dirty and content characteristics. Based on these facts, we present the following expectations on migration’s performance and cost while employing each optimization.

(1) Expectation for delta compression and page skip techniques. For optimizations that account only for page dirty characteristics, we expect the performance improvement to be proportional to the page dirty rate. It is stated in [18] that the additional network traffic generated during migration (which is the difference between total network traffic generated during migration and VM’s memory size) is proportional to the page dirty rate. Hence, we expect the performance improvement to be proportional to the additional network traffic generated with vanilla migration technique.

Figure 1 plots the additional network traffic generated during the 294 migration instances (using vanilla migration technique). Here, the migration instances are sorted in ascending order of additional network traffic and a migration identifier (ID_d) is assigned to each instance. When we employ either delta compression or page skip technique, we expect the performance improvement to increase with increase in the migration ID_d.

(2) Expectation for deduplication technique. For optimization that accounts for both page dirty & content characteristics, we expect the performance improvement to be proportional to both additional network traffic and amount of zero & word level duplicates and total network traffic generated with vanilla migration.

Figure 2 plots the expected amount of zero and duplicate pages present in each of the 294 instances of migration (as determined by analyzing the memory snapshot of the VM). As the deduplication technique is dependent on both dirty and content characteristics, we divided the 294 migration instances into two groups. The migration instances with less than 450 MB of additional network traffic belong to the first group (i.e., from 0 to 190 in Figure 2), whereas all other instances belong to the second group. For both groups, the migration identifier (ID_d) is assigned to each instance by sorting them in ascending order of the sum of zero and duplicate pages.

When we employ the deduplication technique, we expect the performance improvement to be proportional to both zero & duplicate pages and additional network traffic generated. Further, we also expect the number of sub zero pages and sub duplicate pages to be proportional to the number of zero pages and duplicate pages, respectively.

(3) Expectation for data compression technique. Though the compression technique accounts for content characteristics alone, the performance improvement is also dependent on the dirty characteristics. This is because, the opportunity for compression increases with increase in the total network traffic. When we employ data compression, we expect the performance improvement to be proportional to number of zero page & word level duplicates and total network traffic generated with vanilla migration.

(4) Expectation on resource cost. For all optimizations, we expect the CPU utilization at both source and destination server to increase. Figure 3 plots the CPU utilization during migration. With increase in the migration rate, the CPU utilization increased as more pages are transferred and received per second. The CPU utilization at the destination server was five times higher than at the source. The reasons is that, at the source server, the TCP segmentation offload feature was enabled to reduce the CPU overhead of TCP/IP.

3.3 Methodology

We employed the following four steps:

1. With the 294 migration instances, we employ each optimization individually to quantify the improvement in migration time (MT), downtime (DT), and network traffic (NT) over the vanilla migration technique. We also quantify (a) the individual components (refer Table 2) due to which performance gain is observed, and (b) the increase in CPU utilization at the source and destination server.

2. If performance improvement is not as expected for certain migration instances, we identify the application behaviour that explains the anomaly. This helps in establishing a correlation between application behaviour and how well the optimization works.

3. We then compare these optimizations on the basis of resource cost (i.e., total CPU utilization at the source and destination server) and the performance improvement (i.e., total network traffic reduction).

4. Based on our observations from steps 1 and 3, we list the appropriate combinations of optimizations. For these combinations, we again employ step 1 to 3 until we find no more viable combinations.

4. Empirical Evaluation

In this section, we evaluate each optimization individually from the perspectives of performance gain over vanilla migration, resources used to achieve it, and the impact of application behaviour on this performance gain. Table 3 presents a summary of performance gains and cost increases (in terms of additional CPU used to apply the optimization).
Table 3. Performance and cost tradeoff of each optimization, and improvement due to individual component.

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Average reduction due to (in MB)</th>
<th>Average reduction in</th>
<th>Increase in CPU at</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fdp</td>
<td>hdp</td>
<td>del</td>
</tr>
<tr>
<td>Delta Comp.</td>
<td>131</td>
<td>92</td>
<td>54</td>
</tr>
<tr>
<td>Page Skip</td>
<td>289</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Full Page Dedup.</td>
<td>104</td>
<td>33</td>
<td>79</td>
</tr>
<tr>
<td>Sub Page Dedup.</td>
<td>99</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>Data Comp.</td>
<td></td>
<td>62</td>
<td>510</td>
</tr>
</tbody>
</table>

Figure 4. Reduction in the additional network traffic generated while employing delta compression technique.

4.1 Delta Compression

The delta compression technique identifies and transfers only the modified portion of a page instead of the entire page. The size of delta compression cache was set to VM’s memory size to attain maximum performance.

Performance gain. On employing the delta compression technique on 294 migration instances, we observed an average reduction of 17% in the network traffic and 14% in the migration time. The improvement in performance was due to three components namely (i) false dirty pages—fdp, (ii) page delta—del, and (iii) reduction in total number of pages dirtied—dd. The reduction in dirtied pages was observed because of reduction in the iteration time caused by other two components, and we know from [18] that the number of pages dirtied is proportional to the iteration time. For certain workloads, the total number of pages dirtied increased as more number of iterations were executed (which means the enough progress between iterations were observed).

Impact of Application Behaviour. Every dirtied page is either a fdp or eligible for del but not both. This is because, if a page is dirtied after its transmission in an iteration (irrespective of its state before the transmission), it is definitely not a false dirty page but eligible for page delta depending on the modified portion of that page. The main factor which decided the improvement due to fdp and del was page dirty frequency. Figure 4 plots the improvement due to optimization components for each migration instance along with dirty frequency factor ($d_f$) associated with the workload. The value of $d_f$ for a workload was calculated as

$$d_f = \frac{\sum_{i \in \text{dirty_frequency}(\text{page}_i)} - \#\text{pages dirtied once}}{\text{total } \#\text{pages dirtied}}$$

where, $i$ denotes a page. The dirty_frequency(page) denotes the occurrence of dirty bit for page $i$ in the collected bitmaps. As expected, the reduction in network traffic increased with increase in ID$_d$ except for a few workloads. When the dirty frequency factor was low, improvement due to fdp was higher than del (refer to ID$_d$ 190 to 220 in Figure 4). This is because, when a page is dirtied before its transmission in an iteration, the chance for the page getting dirtied again before the next iteration is low (when $d_f$ is low).

On the other hand, when the page dirty frequency was high, the number of pages eligible for del was higher (ID$_d$ 220 to 230, 260 to 267, 275 to 285). However, the improvement due to del was dependent on the size of modified portion of a dirtied page. Specifically, the improvement was low when pages were dirtied due to

1. frequent execution of malloc(), memset(), and free() operations which modified entire page—observed with minebench, parsec, and dacapo benchmarks,
2. disk read (as entire page was modified)—observed with vips and file server workload.

(a) source CPU cost  
(b) destination CPU cost

Figure 5. CPU utilization at source and destination server during migration with delta compression technique.
With a moderate value of dirty frequency factor, the improvement due to \(\text{fdp}\) and \(\text{del}\) was approximately equal for most of the workloads while for a few workloads, \(\text{fdp}\) was little higher than \(\text{del}\).

**Resource cost.** The CPU and memory utilization are the two resource costs associated with delta compression technique. Figure 5 plots the CPU utilization at both source and destination server against the migration rate and performance gain. With increase in the performance gain (due to \(\text{fdp}\) & \(\text{del}\)), the CPU utilization at the source increased. This is because, when the reduction due to \(\text{fdp}\) and \(\text{del}\) was high, the amount of data to be transferred reduced and hence the number of pages processed per second increased (due to decrease in the network I/O waiting time). Similar behaviour was observed with CPU utilization at the destination server.

Further, the processing time of pages were varied depending on the content, delta size, run length encoding, & number of overflows, and hence, some spikes in CPU utilization was observed. The reduction due to \(\text{fdp}\) did not impact the destination’s CPU utilization as the destination had to work only on \(\text{del}\). On average, the CPU utilization at source and destination with the delta compression technique was \(3.2\times\) and \(1.2\times\) higher than the vanilla migration’s CPU utilization, respectively (refer to Figure 3 and 5).

**Impact of Cache Size and Replacement Policy.** In existing works, the cache replacement policy is set to first in first out (FIFO). Figure 6(a) plots the CDF of reduction in the network traffic while using FIFO replacemen policy and four different cache sizes. Full implies that the cache is as large as the VM’s memory size and the other three sizes were half, quarter and eighth of the VM’s memory size, respectively. With decrease in the cache size, the performance gain decreased due to the low hit rate as expected.

We found out that the **FIFO cache replacement policy is not suitable for delta compression as it can retain pages which have zero or low dirty frequency while evicting important pages** (i.e., pages with high dirty frequency). Hence, we propose a new replacement policy called \(\text{fdp-of}\). In \(\text{fdp-of}\), the pages to be evicted are the ones which were either used to identify a false dirty page (means low dirty frequency) or the ones which resulted in overflow—of (while employing XOR and run length encoding) as most portion of pages were modified. When there is no such suitable page for eviction, we employ FIFO policy. Figure 6(b) plots the CDF of reduction in the network traffic while using \(\text{fdp-of}\) replacement policy. As expected, the performance gain improved over FIFO policy.

### 4.2 Page Skip

The page skip technique avoids transferring frequently dirtied pages to reduce the network traffic.

**Performance gain.** On employing the page skip technique on 294 migration instances, we observed an average reduction of 20% in both the network traffic and migration time. If a page gets dirtied before its transmission, it can be either a \(\text{fdp}\) or a \(\text{hdp}\). However, the improvement due to individual components cannot be determined due to the nature of page skip technique and dirty tracking mechanism.

The traffic reduction due to \(\text{fdp}+\text{hdp}\) was significantly higher than \(\text{dd}\) (refer Table 3). Figure 7 plots each component due to which the reduction in network traffic was observed. The reduction due to \(\text{dd}\) was on average 2.6 times higher compared to the delta compression due to execution of more iterations for 48 instances. The downtime with skip technique was on average 1.27 times higher than delta compression. The reason is that the skip technique is not applicable for the stop-and-copy phase unlike delta compression.

**Impact of application behaviour.** As shown in Figure 7, performance gain increased with increase in the migration ID except for a few migration instances (for e.g., IDd 284, 293, 294). The reason for poor performance gain with the certain workloads was that most pages were dirtied only after their transmission in an iteration, and hence the skip technique did not identify these pages. This behaviour was observed with migration instances where the page dirty rate was much lower than the migration rate. This is because, when pages were transferred rapidly, most pages were dirtied only after their transmission due to low page dirty rate. With file server workload, irrespective of the migration rate, performance improvement was low due to very low page dirty rate. Further, the writable working set size \(s\), which is the total number of unique pages that can be dirtied, was

![](Figure7.png)

**Figure 7.** Reduction in the additional network traffic generated due to the page skip technique over vanilla live migration.

![](Figure6.png)

**Figure 6.** Impact of cache size and replacement policy.
equal to the VM size. On the other hand, when either the difference between page dirty rate and migration rate was low or the page dirty was large (irrespective of the migration rate), significant performance gain was achieved.

Resource cost and Impact of parameters. The additional CPU utilization with page skip is negligible as the only overhead is dirty bitmap collection. The dirty bitmap collection frequency (i.e., after transferring every m pages) is the only parameter associated with the page skip technique. We found the value of 1024 to be suitable for m. When the value of m was lower than 1024, CPU utilization increased a bit with negligible performance gain. When the value was greater than 1024, the performance gain reduced.

4.3 Deduplication
The deduplication technique identifies duplicate pages and transfers only one copy of that page to the destination. We can employ this technique either for full pages or sub pages.

Performance gain. On employing this technique for full pages (of size 4 KB) and sub pages (of size 1 KB) on each of the 294 migration instances, we observed an average reduction of 17% and 20% in the network traffic, respectively. The improvement with sub page deduplication was higher due to the additional components such as s2zp, s4dup, and d4el.

Impact of application behaviour. The performance gain due to each component for both techniques are plotted in Figure 8 and 9. As expected, the performance gain due to zp and dup increased with increase in the migration ID of the page (in both groups). As expected, the improvement due to s2zp and s4dup was proportional to the amount of zp and dup. The fdp was also proportional to the additional network traffic except for a few migration instances (for e.g., ID_d 220 to 230 & 270 to 290). This is because of the high page dirty rate.

Resource cost & Impact of parameters. Compared to the vanilla live migration, the CPU utilization for full page deduplication was on average 11 times higher at the source server (due to hash computation and hash comparison), whereas there was no difference in the destination’s CPU utilization. In the case of sub page deduplication, hashing is done on parts of a page and hence the CPU utilization increased (13× higher at the source server).

The sub page size is the only parameter associated with this optimization. We repeated all experiments with a sub page size of 2 KB and 512 bytes. On average, the network reduction with 2 KB and 512 bytes sub page was 2% lower and 2% higher, respectively, as compared to the sub page size of 1 KB. For the sub page size lesser than 1 KB, the migration rate did not go beyond 500 Mbps due to high CPU blocking time with frequent hash computation).

4.4 Data Compression
The data compression technique exploits word level duplicates to improve the migration performance. On employing the data compression technique on 294 migration instances, we observed an average reduction of 37% in both network traffic and migration time. Figure 10 plots dd and vld+zp+s2zp due to which the reduction in network traffic was observed. As expected, the performance gain was proportional to the total network traffic generated with the vanilla live migration except for the file server workload (refer Figure 10). This is because, text files used with the file server workload were created using dd if=/dev/urandom which did not introduce word level duplicates.
Figure 10. Reduction in the additional network traffic generated due to data compression technique over vanilla live migration.

Figure 11. CPU utilization at source and destination server during migration with data compression technique.

Figure 12. Network traffic reduction.

Figure 13. Total CPU utilization during migration.

Performance and cost tradeoff. Figure 12(a), and Figure 13 plot the CDF of the total network traffic reduction and total CPU utilization for each optimization, respectively. Out of these five optimizations, page skip and data compression techniques reduced the total network traffic significantly while utilizing very little CPU at both source and destination server. The total CPU utilization at the source for both delta and data compression were approximately equal even when the CPU utilization per second with data compression was nearly twice as compared to delta compression (as the reduction in network traffic was twice). With deduplication, the total CPU utilized was significantly larger though the reduction in total network traffic was small.

Table 3 summarizes the average improvement due to each individual component. Out of nine components, the improvement due to fdp was higher than any other components. This is because, almost all workloads have pages which are dirtied less frequently. Figure 12(b) plots the CDF of traffic reduction only due to fdp and hdp. Page skip is the only optimization which identified both fdp and hdp while utilizing same amount of CPU as vanilla live migration.

Appropriate combination of optimizations. Every dirtied page is either a fdp or a hdp and the page skip technique identifies both. Further, the improvement due to fdp+hdp is high as compared to all other components (refer to Table 3). Therefore, we conclude that the page skip technique should be employed irrespective of the application behavior. Only in addition to the page skip technique, we should employ either deduplication or delta or data compression. Specifically, for unskipped pages during each iteration, we can employ one of the other optimizations.

Figure 14 plots the percentage of pages, which were dirtied during an iteration, got skipped, and the amount of pages

4.5 Comparison of Optimizations

In previous sections, we presented the results of evaluating each optimization in terms of performance gain and cost increases over the vanilla pre-copy migration. In this section, we compare optimizations with each other on the basis of total network traffic reduction and total CPU utilization to find the best optimization in terms of performance & cost. Further, this comparison will aid us to find the appropriate combinations of optimizations.
which were not skipped for both iteration 1 and other iterations. The percentage of pages skipped in the first iteration and other iterations was less than 50% for only 9% and 4% of 294 migration instances, respectively. The amount of pages not skipped in the iteration 1 was higher than other iterations (as all pages were scheduled to transfer in the iteration 1 as compared to only dirtied pages for other iterations).

The delta compression technique significantly reduced the downtime as compared to all other optimizations (refer Table 3). Hence, employing this optimization in addition to the page skip technique will reduce the downtime even more. Further, the data compression technique can be employed for every page before its transfer as it reduces the network traffic significantly. Zero content present in the VM is also handled by data compression. Hence, we need to apply deduplication very selectively by analyzing the expected amount of duplicate pages (but not zero pages) present in the VM as the cost in terms of CPU utilization will be high. Further, when page skip is the base optimization, we expect the impact of application behaviour on performance gain to change and are summarized in Table 4.

Based on these reasonings, we recommend the following three combinations of optimizations—with page skip as base optimization, employ either delta compression or deduplication or data compression.

5. Evaluation of Combinations of Optimizations

We employ each combination of optimizations on the 294 migration instances and present the performance gain and cost over vanilla live migration. First, we evaluate combinations in which page skip is the base optimization. Based on this evaluation, next we find other suitable combinations of optimizations that can be employed. Table 4 summarizes the performance and cost associated with various combinations of optimizations over the vanilla live migration.

5.1 Page Skip as Base Optimization

We employ delta compression, deduplication, and data compression individually along with page skip to quantify the effect of combined optimizations on the performance & cost, and also to study the impact of application behaviour.

(1) Page Skip with Delta Compression. On employing both page skip and delta compression techniques together, the average reduction in network traffic was 21.4% which was just 1.4% increment over page skip optimization (when applied alone). However, the average reduction in migration time was 4% higher as compared to the page skip optimization (when applied alone). This is because of small reduction in the CPU utilization which decreased CPU waiting time (.3x, refer to Table 3 and 5). The improvement due to fdp+hdp reduced as compared to page skip optimization because of increment in dd. As shown in Figure 14 with increase in the migration IDd, the reduction in network traffic increased except for file server workload because of low page dirty rate (& large writable working set size which was equal to VM size) and high disk reads.

(2) Page Skip with Deduplication. On employing both page skip and deduplication techniques together, the average reduction in network traffic was 27% with full page deduplication (7% increment over only page skip) and 29% with
Table 5. With page skip as base optimization, performance and cost trade-off of each combination over vanilla.

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Average reduction due to (in MB)</th>
<th>Average reduction in</th>
<th>Increase in CPU at</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fdp</td>
<td>hdp</td>
<td>del</td>
</tr>
<tr>
<td>Delta Comp.</td>
<td>277</td>
<td>43</td>
<td>32</td>
</tr>
<tr>
<td>Full Page Dedup.</td>
<td>275</td>
<td>32</td>
<td>69</td>
</tr>
<tr>
<td>Sub Page Dedup.</td>
<td>270</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>Data Comp.</td>
<td>238</td>
<td>71</td>
<td>377</td>
</tr>
</tbody>
</table>

Figure 16. Reduction in the network traffic generated while employing both page skip and sub page deduplication.

Figure 17. Reduction in the network traffic generated while employing both page skip and data compression.

sub page deduplication (9% increment over using just page skip). The CPU cost associated with deduplication technique was still high. It is advisable to avoid sub page deduplication as the average reduction in network traffic was only 2% higher than full page deduplication while doubling the CPU cost. Figure 16 plots the reduction in network traffic due to each component. With the file server workload, the reduction in network traffic was less due to a smaller page dirty rate and lower zero & duplicate content. For a few migration instances (between ID_d 199 and 249), the reduction in network traffic was large due to the copious amount of zero and duplicate content in the VM.

(3) Page Skip with Data Compression. On employing both page skip and data compression techniques together, the average reduction in network traffic was 41% which was just 4% increment over data compression but 21% increment over page skip. However, the CPU utilization (per second) at the source and destination did not reduce significantly. The improvement due to fdp+hdp reduced significantly due to increase in dd. The reduction in dd was observed due to reduction in the iteration time. Figure 7 plots the reduction in network traffic due to each component. Other than file server workload, significant reduction was observed.

Total CPU utilization with combined optimizations. Figure 18 plots the CDF of total CPU utilization at both source and destination server during migration when page skip was the base optimization. The total CPU utilization at both source and destination decreased due to combination of optimizations (refer to Figure 13 and 18). For e.g., the average reduction in total CPU utilization at the source for page skip and data compression combination was 60%.

The combination of page skip and data compression is the best in terms of performance and cost. Hence, in the next section, we consider page skip and data compression as the base optimizations and employ either delta compression or deduplication technique.

5.2 Page Skip and Data Compression as Base

(1) Page Skip and Data Compression with Delta Compression. On employing delta compression technique along
with page skip and data compression, the average reduction in network traffic increased only by 3% and there was no reduction in migration time (over page skip and data compression optimizations). Further, the CPU utilization per second also increased and became the sum of CPU utilized by combination of (i) page skip & delta, and (ii) page skip & compression. This is because, when a page belongs to \( \delta_1 \), we did not employ data compression (as run length encoding was already employed by the delta compression).

(2) Page Skip and Data Compression with Deduplication. On employing deduplication technique along with page skip and data compression, the average reduction in network traffic increased only by 2% with full page deduplication. With sub page (of size 1 KB), the performance improvement got worsened (3% increase in the additional traffic) due to decreased migration rate (because of huge contention on CPU). Further, the CPU utilization per second also became the sum of the cost of individual optimization. This shows that any more combinations of optimizations is not useful.

Hence, we conclude that the page skip, and a combination of page skip and data compression are more appropriate.

6. Guidelines for Employing Optimizations

In this section, we present a list of guidelines on how to use these optimizations for various migration scenarios.

1. The page skip is the predominant optimization in terms of a performance and cost tradeoff. It would be the preferred optimization to apply first under any circumstance.
2. The deduplication technique must be applied with utmost care as the CPU cost associated with this technique is high. In our study, we estimated zero and duplicate content using memory snapshot of the VM. However, this may not be applicable for live data center applications. It is advisable to use a memory sharing tool such as KSM to estimate the amount of zero & duplication content, and then make the decision whether to employ this technique.
3. Determining whether to apply data compression is dependent on knowing the extent of word level duplicates in a page. This can often be difficult to estimate.
4. If reducing the downtime is main goal, delta compression is the best technique. However, if memory bandwidth or memory size is the bottleneck, this technique will worsen the situation even further. This is because, if the cache size is too small, delta compression will not result in any performance gain. The size of the cache should be equal to the maximum writable working set size [17] to achieve the maximum performance.
5. The combination of optimizations (with page skip as the base) reduces total CPU utilization as well as utilization per second at the source. Hence, this would be very useful while migration is employed to resolve resource hotspot mitigation.
6. If resource hotspot is due to the network bandwidth, it is advisable to employ both page skip and data compression technique as this combination reduces network traffic immensely while consuming a moderate amount of CPU.

7. Conclusion and Future Work

In this paper, we performed a comprehensive empirical study to understand the performance/cost trade-off of various optimizations for VM live migration. Specifically, we found the page skip technique to be more efficient compared to all other optimizations. Further, we correlated application behaviour such as page dirty frequency factor, maximum writable working set size, page content similarity to performance improvement of each optimization. Further, we provided recommendation on how to combine various optimization such that the performance is improved while paying lesser CPU cost (compared to individual optimization).

As a part of future work, we would like to propose an analytical model for each of these optimization which takes into account of application behaviour and resource allocation for the migration process. Such a model will be useful to take precise decision on certain migration scenario.

A. Workloads

The following are workloads used in this paper:

Web Services: (a) HTTP File Server—where clients download files at two different rates of 80 Mbps and 160 Mbps (b) RUBiS [21]—an auction site prototype modeled after eBay.com which implements the following features: browse and bid on existing items, register and sell items. (c) Mediawiki [15]—an open source wiki package written in PHP, mainly used in all Wikipedia websites. The workloads (a), (b) and (c) are two tier applications, i.e., web server and a database server. The Apache Jmeter [7] was used to generate load for RUBiS and Mediawiki. For RUBiS, the Jmeter was configured with 50, 100, and 300 clients in three different runs, respectively. For mediawiki, the Jmeter was configured with 10 and 20 clients in two different runs, respectively.

OLTPBenchmarks [6]—an open source testbed for benchmarking database management systems. It implements different set of workloads such as yesh, twitter, seats, votes, lpec, and tatp.

Multimedia, Data Mining and Multi-threaded Benchmarks: (a) Parsec [2]—a benchmark suite that consist of multi-threaded programs from different area such as computer vision, image and video processing, and animation physics. We used the following six applications: bodytrack, ferret, fluidanimate, freqmine, vips, and x264. (b) NU-MineBench [16]—a data mining workload which implements many mining algorithms such as ECLAT, HOP, ScalParC, and UtilityMine. (c) Dacapo [3]—an open source client-side JAVA benchmark suite consist of avoror, eclipse, fop, h2, jython, luindex, lusearch, pmd, sunflow, tomcat, tradebbeans, tradesoap, and xalan. (d) Kernel compile—which compiles Linux kernel v2.6.39 with the default configuration.
References


