width between the physical hosts. Although the VMScatter+CM mode consumes more time, it enhances the VMScatter further by 70.6%, and achieves a total of 92.4% reduction over the off-the-shelf method.

As Figure 8(b) shows, the increase of total transferred data in VMScatter is not proportional to the number of VMs. This is because the inter-identical pages are only transferred by multicasting a single copy, so the identical pages in the new added VM will not need to be transferred any more except extra page references such as *Page Address* and *VM Id* pairs. The increased amount are mainly from the unique pages in the added virtual machines as well as the additional dirtied pages caused by longer migration time.

It should be observed in the Figures 9(a) and 9(b): both the total migration time and total transferred data remain unchanged regardless of the number of target hosts. The reasons are explained as follows. 1) The definitive identical rate of 12 VMs as shown in Figure 6 implies that the amount of packets is almost fixed for both identical and unique pages, which further indicates the transfer time for these two types of packets is definitive in limited network bandwidth. As a result, the amount of dirtied pages can be regarded as fixed. In addition, the lost packets increase the transferred data but only a small number (only about 0.3%). Consequently, the amount of total transferred data consists of the above three types of packets and can be considered to be constant. 2) For the total migration time, the time spent on the preparation phase including collecting identical pages and scheduling groups is almost constant for fixed page numbers, therefore the total migration time is in line with the page transfer time, thus it is also constant.

Network traffic. Although there is no exact method to quantify network traffic during the live migration, we provide an approximate measure by the sum of packets received by target hosts. Figures 8(c) and 9(c) compare the network traffic with the increasing number of VMs and target hosts respectively. The network traffic is equal to the total transferred data when the number of targets is one, this is easily understood by the way we measure the network traffic. Another result to be observed is that when three (12) VMs are migrated to three (12) target hosts as shown in Figure 8(c) (9(c)), i.e., each target host receives only one VM, the VMScatter method still reduces the network traffic by 17.8% attributable to the self-identical pages within the VM. For other scenarios, the network traffic in VMScatter mode decreases significantly with a range between 50.1% to 70.3%.

Different from Figure 9(b) where the total transferred data are constant over various number of hosts, the network traffic increases as the number of target hosts increase as illustrated in Figure 9(c). This is because one additional copy of the packets needs to be forwarded by the switcher to the new added target host during multicast over the network. The VMScatter+CM mode also gain performance, reducing the multicast traffic further by about 69.7%.

We also evaluate the three metrics under Sysbench and TPC-W workloads. For the TPC-W which has lower identical rate, the VMScatter live migration method still performs nicely by reducing 63.3% of the total migration time, 67.4% of the total transferred data and 55.8% of the network traffic.

Overall, these results confirm the effectiveness of VMScatter. Although the compression and multithreading method produces longer total migration time, it reduces numerous transferred data and network traffic further by about 70% on the basis of VMScatter mode.

4.4 Downtime

Downtime is another important metric of live migration. It consists of the time to suspend the VM at the source host, transfer the dirtied pages, and activate the migrated VM at the target host. The downtime is inevitable because the dirtied pages generated during continuous data transfer will lead to the inconsistency of VM state between the source and target host.

Table 1 shows the comparison in terms of downtime for the three modes for migrating 12 VMs to three targets evenly. The variation in the downtime is due to the parallel migration. The VMScatter mode performs better than the off-the-shelf method, and this could because this mode generates less dirtied data in a shorter migration time, thus consumes less time in the final data transfer after suspending the VM. Consider the VMScatter+CM mode, the overhead of compression at the source and decompression at the target cause the minimum value to be larger than the other two modes, and the average is less than off-the-shelf due to lesser time to transfer the reduced packet size.

4.5 Grouping Benefit

The two most significant results we have seen so far are in Figures 8 and 9 where the total transferred data and network traffic are reduced. We then conduct experiments to evaluate our grouping method which aims to reduce the network traffic further by deciding a preferable placement.

Figure 9(c) demonstrates the variation of network traffic with different groupings when the number of target hosts varies. Furthermore, we fix the number of targets and construct a group that distributes twelve VMs evenly to each target, we distribute evenly for fair comparison since the difference in number may result in volatile identical rate which affects the results. For each fixed number of targets, we simulate 60 different groupings, migrate the 12 VMs to the associated target hosts decided by each grouping and then count the network traffic. Besides, we obtain the preferable grouping by the greedy algorithm. We set the capacity of target hosts as identical, which means the hosts will accept the same number of VMs.

Intuitively, there is only one grouping method when there is one host, where all the VMs would target one host; the same is true for 12 targets where each VM targets a respective host. Thus, the two scenarios are not our concern. Table 2 illustrates the results of the network traffic on different groupings under various workloads, including the maximum value, minimum value and average value of the network traffic on 60 groupings; along with the network traffic of our preferable grouping. As the table shows, the maximum traffic is 4.07G while the minimum value is 3.47G for targeting three hosts when Kernel Compilation running inside the VMs, and the difference between the two groupings is about 17.3%. Our preferable placement of VMs decreases the network traffic to 3.31G, a 13.4% reduction compared to the average value. Generally, the preferable grouping achieves 10% to 15% reduction of the network traffic against the average of random groupings, thus it proves the improvement of grouping algorithm. The 10% to 15% reduction of network traffic is particularly valuable in the data-intensive data centers.

4.6 Performance Impact

In this section, we quantify the side effects of migration on a couple of sample applications. We evaluate the performance impact on both single VM and VM cluster with migrating 12 VMs to

Modes	**Max**	**Min**	**Avg.**
Off-the-shelf	2351	192	1518
VMScatter	1573	184	863
VMScatter+CM	1483	576	1132

Table 1. Comparison of downtime (ms).

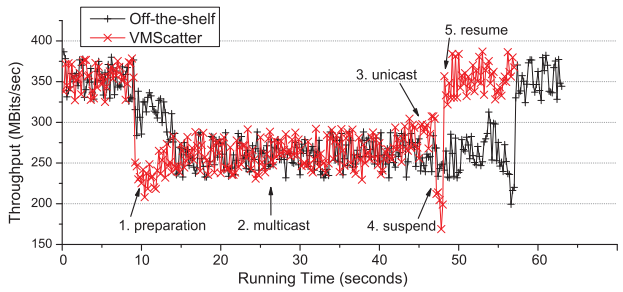

Figure 10. Throughput during live migration (result of off-the-shelf mode has been truncated to save space).

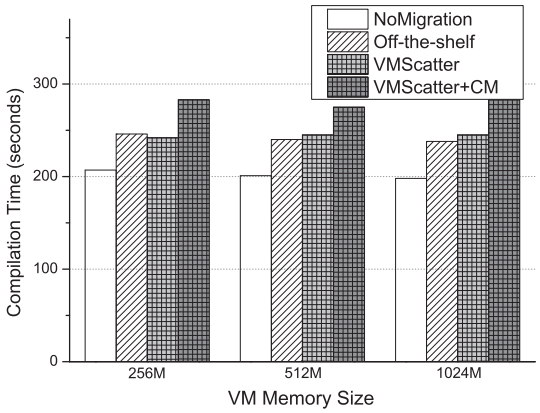

Figure 11. Compilation time on migration.

three targets, and illustrate the results of off-the-shelf mode versus VMScatter mode.

Impact on Single VM. We start first by measuring the performance on a single VM in terms of throughput per second by examining the live migration of a Apache web server serving static content at a high rate. The web server served 1000 files of 512 K-Bytes, all of which were cached on memory. In this experiment, 10 httperf processes in a remote client host sent requests to the server in parallel. Figure 10 illustrates the throughput achieved when continuously serving concurrent clients. At the start of the trace, the normal running of the VM can serve about 354Mbits/sec. After the live migration starts at the 9th second, the throughput of VMScatter decreases to about 214Mbits/sec which is lower than 309Mbits/sec of off-the-shelf. This is because the collecting and scheduling stages consume more CPU resource than off-the-shelf mode which only set flags. Then the page transfer phase serves 272Mbits/sec for about 24 seconds. There is no obvious decrease compared to off-the-shelf mode, thus implying the optimized permutation of packets takes effect. The throughput in transferring dirtied pages keeps about 305Mbits/sec which is higher than multicast. This may be because the amount of dirtied pages in unicast is less than the amount of pages during multicast, thus reserving more CPU and network resource for applications running insides VMs. One sudden decrease is the result of VM suspending. After the VM is resumed at the target host, the throughput returns to normal.

Impact on VM Cluster. We evaluate the performance of VMScatter on VM Cluster via *distcc* [1] to build a kernel compilation cluster to distribute the compilation tasks across the 12 VMs, and migrate back and forth repeatedly between the source and three target hosts. Figure 11 compares the completion time for various memory size of VM under three live migration modes, the result without migration is also given for comparison. The VMScatter mode consumes almost the same compilation time as the off-the-shelf method, and both increase by less than 20% compared to NoMigration, owing to the similar CPU and network utilization. The VMScatter+VM mode cost more time because the CPU overhead of compression at the source host and decompression at the target hosts.

5. Related Work

Live Migration. Clark et al. [10] first propose the pre-copy live migration based on Xen platform, they transfer the page iteratively, and boot the VM when the consistent state are reserved in target host. However pre-copy migration may fail in harsh scenes such as low network bandwidth and memory-intensive workload where the amount of dirtied pages cannot converge. Hines et al. [27] propose post-copy method, they first boot the VM on target host and then copy the pages on demand, thus the memory pages will be transferred only once which both solves the problem of pre-copy and reduces the total transferred data. Liu et al. [19] adopt the methods of ReVirt [12], achieve live migration by transferring the log which records the execution of VM and replaying them at target host. Deshpande et al. [11] consider migrating multiple machines from one host to another, and propose live gang migration by page sharing and delta transfer to reduce the amount of transmission. In contrast, our concern is with the one-to-many migration and implementing VMScatter by multicasting a single copy of the identical pages instead of individual transfer.

Multicast. Multicast has been used to transfer images or snapshots for deploying multiple identical VMs in the IaaS platform [18, 26]. VMScatter employs the multicast method to transfer the identical pages by single copy, combined with unicast to transfer the unique pages and dirtied page. Besides, we specify a permutation of packets with the solution of Hamilton Cycle problem to reduce the network overhead occurred in multicast groups' switchovers.

Page Sharing and De-duplication. Page sharing saves memory consumption of VMs by merging the identical pages into one physical page. Bugnion et al. propose Disco [9], a tool that uses transparent page sharing to de-duplicate the redundant copies across VMs. Waldspurger et al. [28] improve Disco further by content-based page sharing. Milos et al. [21] use sharing-aware block devices for detecting duplicate pages on Xen virtual machine monitor. Gupta et al. [14] improve the page sharing rate among VMs by dividing

Target Count	2				3				4				6			
Benchmarks	**Max**	**Min**	**Avg.**	**Prefer.**	**Max**	**Min**	**Avg.**	**Prefer.**	**Max**	**Min**	**Avg.**	**Prefer.**	**Max**	**Min**	**Avg.**	**Prefer.**
Compilation	3.56	3.12	3.37	3.05	4.07	3.47	3.82	3.31	4.86	4.11	4.65	4.03	5.97	5.12	5.73	5.18
Sysbench	3.82	3.33	3.67	3.32	4.22	3.69	4.07	3.52	5.08	4.33	4.86	4.30	6.23	5.34	6.02	5.45
TPC-W	4.84	4.37	4.60	4.34	5.33	4.71	5.15	4.89	6.17	5.41	5.98	5.49	7.29	6.86	7.15	6.89

Table 2. Comparison of network traffic(GBytes) for groupings, the target host count is 2, 3, 4, 6.

the page to sub-pages. Arcangeli et al. proposes KSM [5], a kernel module in Linux that uses an unstable red-black tree to improve the efficiency. We share a similar philosophy to page sharing, but against the motivation of page sharing that focus on less physical memory consumption, VMScatter is interested in de-duplicating the identical pages in the packet to be multicasted. Furthermore, our approach combines the selective hash with the red-black tree, and achieves an order of magnitude speedup over the original hash method on organizing millions of memory pages.

Placement of VMs. Many works have adopted live migration technology to achieve power saving [23], load balance [29], SLA [6], quality of service (QoS)[24], etc. In this paper, we consider the network traffic metric and propose a grouping algorithm with the aim of minimizing network traffic by selecting a preferable placement of VMs.

6. Conclusions

We implemented VMScatter to migrate VMs to multiple hosts. Our design and implementation addressed the issues involved in live one-to-many migration, placement of VMs and multicast specific options. By merging the identical pages into one page, VMScatter multicasts the single page to many targets instead of transferring these pages individually. The novel grouping method guides the VM's destination with respect to the network traffic over the network. And we explore a further benefit allowed by compression and multithreading. Through detailed evaluation, we show that the performance is sufficient to make VMScatter a practical tool in data centers even for VMs running interactive loads. In the future, we plan to investigate providing disk state migration, perhaps using existing techniques to improve VMScatter for hosts connected to independent storage, and evaluate VMScatter in complex network topologies such as BCube [13].

Acknowledgments

We acknowledge Yang Cao for his contributions to the algorithm of this work and Bin Shi, Kun Liu for the experimental setup. We also thank the anonymous reviewers for their valuable comments and help in improving this paper. This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302602, National High Technology Research 863 Program of China under Grant No. 2011AA01A202, National Nature Science Foundation of China under Grant No. 61272165, No. 60903149 and No. 91118008, and New Century Excellent Talents in University 2010 and Beijing New-Star R&D Program under Grant No. 2010B010.

References

[1] Distcc. `http://code.google.com/p/distcc/`.

[2] Superfasthash. `http://www.azillionmonkeys.com/qed/hash.html`.

[3] Sysbench. `http://sysbench.sourceforge.net/`.

[4] Tpc-w. `http://www.tpc.org/tpcw/`.

[5] A. Arcangeli, I. Eidus, and C. Wright. Increasing memory density by using ksm. In *Proceedings of the linux symposium*, pages 19–28, 2009.

[6] N. Boboroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, 2007.

[7] B. Bollobas, T. I. Fenner, and A. M. Frieze. An algorithm for finding hamilton paths and cycles in random graphs. *Combinatorica*, 7(4): 327–341, 1987.

[8] M. S. Borella, D. Swider, U. S, and B. G.B. Internet packet loss: Measurement and implications for end-to-end qos. In *Proceedings of ICPP Workshps*, pages 3–12, 1998.

[9] E. Bugnion, S. Devine, Kinshuk, Govil, and M. Rosenblum. Disco: running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4):412–447, 1997.

[10] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of NSDI*, pages 273–286, 2005.

[11] U. Deshpande, X. Wang, and K. Gopalan. Live gang migration of virtual machines. In *Proceedings of HPDC*, pages 135–146, 2011.

[12] G. W. Dunlap, S. T. Kin, S. Cinar, M. A. Basrai, and P. M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of OSDI*, pages 211–224, 2002.

[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: A high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, pages 63–74, 2009.

[14] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. Difference engine: Harnessing memory redundancy in virtual machines. *Communications of the ACM*, 53(10): 85–93, 2010.

[15] H. Jin, L. Deng, and S. Wu. Live virtual machine migration with adaptive memory compression. In *Proceedings of CLUSTER*, pages 1–10, 2009.

[16] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. Kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, pages 225–230, 2007.

[17] M. J. Knieser, F. G. Wolff, C. A. Papachristou, D. J. Weyer, and D. R. McIntyre. A technique for high ratio lzw compression. In *Design, Automation & Test in Europe*, pages 10–16, 2003.

[18] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannel, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *Proceedings of EuroSys*, pages 1–12, 2009.

[19] H. Liu, H. Jin, and X. Liao. Live migration of virtual machine based on full system trace and replay. In *Proceedings of HPDC*, pages 101–110, 2009.

[20] R. E. Miller and J. W. Thatcher, editors. *Complexity of Computer Computations*. Plenum Press., New York, 1972.

[21] G. Milos, D. Murray, S. Hand, and M. A. Fetterman. Satori: Enlightened page sharing. In *USENIX Annual Technical Conference*, pages 1–14, 2009.

[22] S. B. Moon, J. Kurose, P. Skelly, and D. Towsley. Correlation of packet delay and loss in the internet. Technical report, 1998.

[23] R. Nathuji and K. Schwan. Virtualpower: Coordinated power management in virtualized enterprise systems. In *ACM Symposium on Operating Systems Principles*, pages 265–278, 2007.

[24] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: Managing performance interference effects for qos-aware clouds. In *Proceedings of EuroSys*, pages 237–250, 2010.

[25] M. Nelson, B.-H. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *USENIX '05 Technical Program*, 2005.

[26] B. Nicolae, J. Bresnahan, and K. Keahey. Going back and forth: Efficient multideployment and multisnapshotting on clouds. In *Proceedings of HPDC*, pages 147–158, 2011.

[27] M. R, Hines, and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of VEE*, pages 51–60, 2009.

[28] C. A. Waldspurger. Memory resource management in vmware esx server. In *Proceedings of OSDI*, pages 181–194, 2002.

[29] Y. Zhao and W. Huang. Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud. In *Fifth International Joint Conference on INC, IMS and IDC*, pages 170–175, 2009.

[30] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. on Information Theory*, 23(3):337–343, 1997.