Name of system: SnowFlock

Reviewer: Thang Minh Le

Date: November 7th, 2011

Author of software: H. Andrés Lagar-Cavilla

Author of book chapter: Roy Bryant & H. Andrés Lagar-Cavilla

Five star rating of book chapter: ****

Purpose of the system: Cloud computing is attractive for its scalability, high performance and low cost. Taking advantages of hardware virtualization, a cloud service can address any hardware demand starting from a single host to a cluster of thousand hosts. A host in a cloud is a pure virtual machine (VM) rather a physical machine. This makes cloud an ideal environment to run computational intensive applications. However, the issue of most cloud services nowadays is the lack of an efficient way to create a large cluster of hosts. SnowFlock has a solution to this matter. In its simplest form, SnowFlock provides a set of APIs to programmatically create multiple VMs in seconds. Each new VM created by SnowFlock is a cloned VM of the original VM. In order to achieve performance acceleration, SnowFlock has its design drawn from these insights:

- It is possible to start executing a child VM from a minimal set of data
- Children will typically access only a fraction of the original memory image of the parent
- It is common for children to allocate memory after forking
- Children clearly work on a great amount of temporal locality in their memory accesses

Lazy state replication and multicast distribution are the main themes of SnowFlock. On experiments conducted with 128 processors, SnowFlock achieves 7% better compared with optimal execution. It achieves sub-second VM fork irrespective of number of clones. SnowFlock is an order of magnitude faster and sends less than two orders of magnitude than VM fork based on suspend/resume or migration. SnowFlock has proved its performance advance in different load tests including:

- NCBI Blast: the most popular computational tool used by biologist.
- SHRiMP: a tool for aligning large collections of very short DNA sequences against a know genome.
- QuantLib: an open source toolkit widely used in quantitative finance
- Aqsis – Rednderman: an open source implementation of Pixar's Renderman.
- Distcc: a software distributes builds of C/C++ program over the network for parallel compilation.

Basic Metrics

KLOC: http://cloc.sourceforge.net v 1.55 T=31.0 s (174.5 files/s, 49677.7 lines/s)

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                             2051          99077         101556         532271
Bourne Shell                   178          21890          27328         211849
C/C++ Header                  1338          26007          42233         122283
C++                            386          21751          14246          81245
```

```
m4                     43       2905      4527      45734
Java                   298      6160      6165      35844
HTML                   340      1202      461       28033
Python                 335      7897      9099      27544
Assembly               104      2606      7669      23250
make                   201      1626      851       5679
Fortran 77             54       1000      1172      4604
Fortran 90             15       263       466       3517
Teamcenter def         4        20        0         2162
Perl                   11       311       271       1776
Bourne Again Shell     18       223       266       1285
C Shell                18       60        293       874
MUMPS                  1        134       0         793
Expect                 2        0         0         590
XML                    5        42        74        353
CSS                    3        18        5         134
XSD                    1        1         3         115
XSLT                   1        7         2         48
IDL                    1        4         0         44
DOS Batch              1        1         0         38
Javascript             1        4         0         27
DTD                    1        6         2         11
-------------------------------------------------------------------------------
SUM:                5411    193215    216689    1130104
-------------------------------------------------------------------------------
```

Project start-up: the first public release was in September 2008

Number of major release: there has no major release. The product is still in development. The latest release is minor version 2.

Size of user community or number of installations: SnowFlock is released under GPL license. Community mostly is researchers from University of Toronto and Carnegie Mellon University. The size of community is small less than 10.

Major stakeholders: the tool is built for any cloud services using Xen virtualization technology. Major stakeholders are cloud service venders, researchers and any developers who want to get involved.

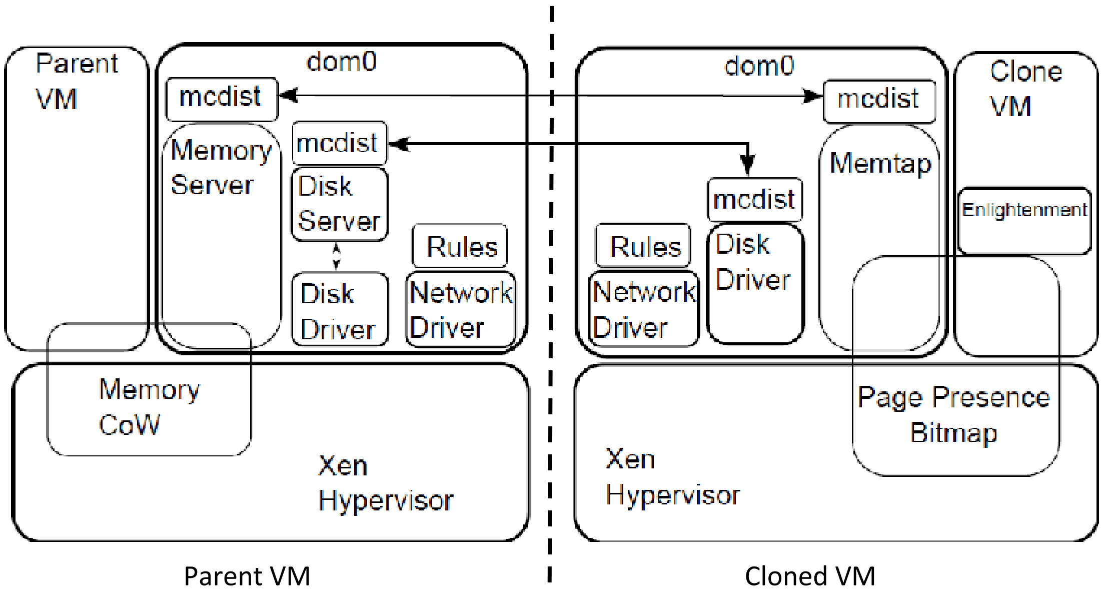
Use of concurrency: Concurrency happens in multicast distribute module where it takes a role of server and multicast requested data to all cloned VMs. Instead of only sending data to the requested cloned VM, it multicasts the data to all clones, which is essentially a pre-fetch mechanism for other cloned VMs who do not make requests for this data.

Implementation language: mainly in C/C++

Supporting software: the SnowFlock implementation is particular for Xen hypervisor. It can only run widely on Unix/Linux system with x86 architecture (also x86_64, IA64, ARM) with Xen.

High Level Architecture

Diagram of software architecture:



memServer: provides all clones with the data they need from the parent. The server sits and waits for page request from clones.

Memory CoW: a copy-on-write mechanism to keep the memory required by cloned VMs unmodified in parent VM

mcdist: distribution service which multicasts data to all cloned VMs. It takes the advantages of network hardware parallelism to efficiently distribute parent states to all cloned VMs.

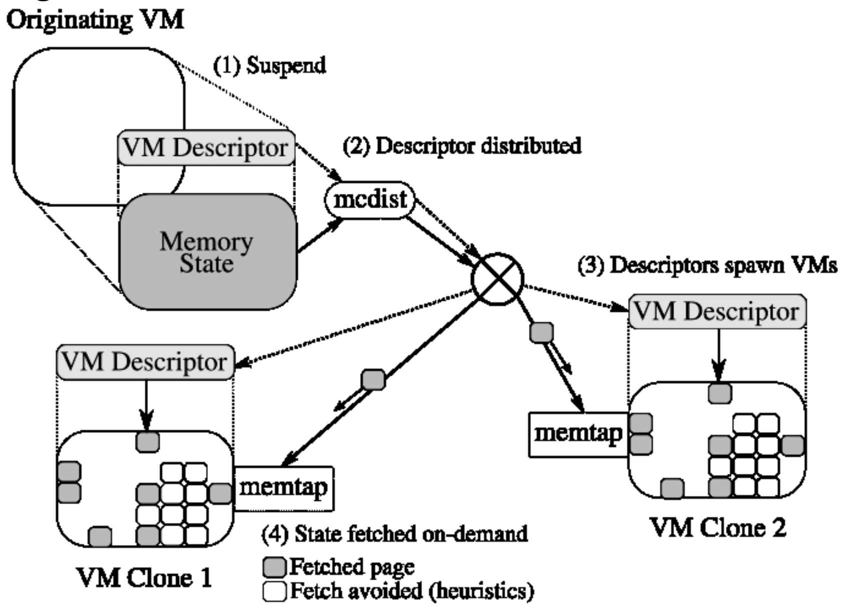
diskServer: most of data accesses of cloned VMs are through suitable distributed filesystems such as HDFS or PVFS, diskServer just keeps binaries and configuration files and hence has similar function with *memServer*

memtap: acts on cloned VM's behalf to communicate with the memory server to request pages that are needed but missing

SnowFlock is architected to adopt client/server replication model. A cloned VM takes a role of a client to send requests to server for fetching missing data. Server replies to the requested client in a multicast manner. The data sent to all other VMs as a pre-fetching advantage. Multicasting data to all clones significantly improves the cloning process. Other three optimizations specific to SnowFlock are:

- LockStep Detection: In the case when multiple clones request the same page in very close succession, mcdist server ignores all but the first of such requests
- Flow Control: To avoid clones to be drowned by too many pages sent by an eager server, the server throttles its sending rate to a weighted average of client's receive rate
- End Game: when the server has sent most pages, it falls back to unicast responses.

High level scenarios:



(1) **Suspend:** SnowFlock suspends the current VM (original VM) to create *VM Descriptor* and *Memory Server*.
(2) **Descriptor Distribution:** SnowFlock uses mcdist to multicast *VM Descriptor* to all clones.
(3) **Descriptor spawn VMs:** after receiving *VM Descriptor*, SnowFlock will initiate clones from the descriptor. At this point, all clones are mostly empty state-wise since *VM Descriptor* only has the most important data to instantiate a new clone only. All other data will be requested by clones when in need.
(4) **State fetched on-demand:** clones send request to parent VM for a require state

Architectural style: SnowFlock adopts object oriented architectural style. Each module has its own role and interacts with other modules for different services. For example, *Memory State* uses *mcdist* to send state data to cloned VMs. *Memory State* is responsible to maintain a frozen coy of parent's state.

Major evolutionary changes: there have been no evolutionary changes since the SnowFlock paper published in 2008.

Performance bottlenecks: As mentioned in the chapter, SnowFlock performs well in most workload. However, the lazy replication approach isn't free. In some workload, it is indeed the bottleneck. The reason is due to during state transfer, the clone is left waiting for state to arrive before it can continue execution. This blocking is somewhat degrades the clone's performance in some cases. For example, a cloned database server.

Real time: the system has real time requirement.

Notation for architecture: INC

Methodology: INC

Appendix

1. Size: L
2. Criticality: Med
3. Age of system: M
4. Rate of change: Lo
5. Business model: open source
6. Stable architecture: Lo
7. Team distribution: Lo
8. Governance: Lo