Instrumentation of Xen VMs for efficient VM scheduling and capacity planning in hybrid clouds.

Kurt Vermeersch
Coordinator: Sam Verboven

Universiteit Antwerpen

Goal

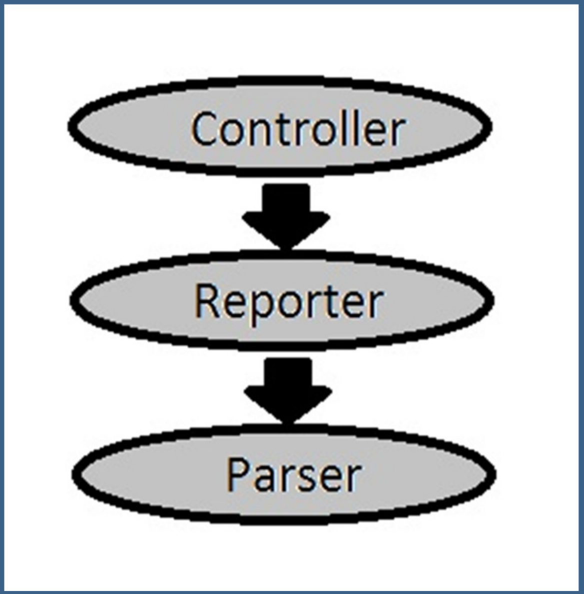
- Phase 1: Extract information about running Xen guests and present it in a clear way.
- Phase 2: Mapping of a VM to a profile used by cloud providers.

Background

- Importance of virtualization today due to the advantages of server consolidation (e.g. cost reduction).
- Virtualization in itself does not guarantee efficiency, several system-level performance metrics should be monitored.
- This data should be used to make intelligent scheduling decissions.

XenBench

- Startup
  - Initialize app
  - Stress system
- Gathering
  - Start profiling tools
- Stopping
  - Write unaltered output to file
- Analysation
  - Parse files & present info in a clear way



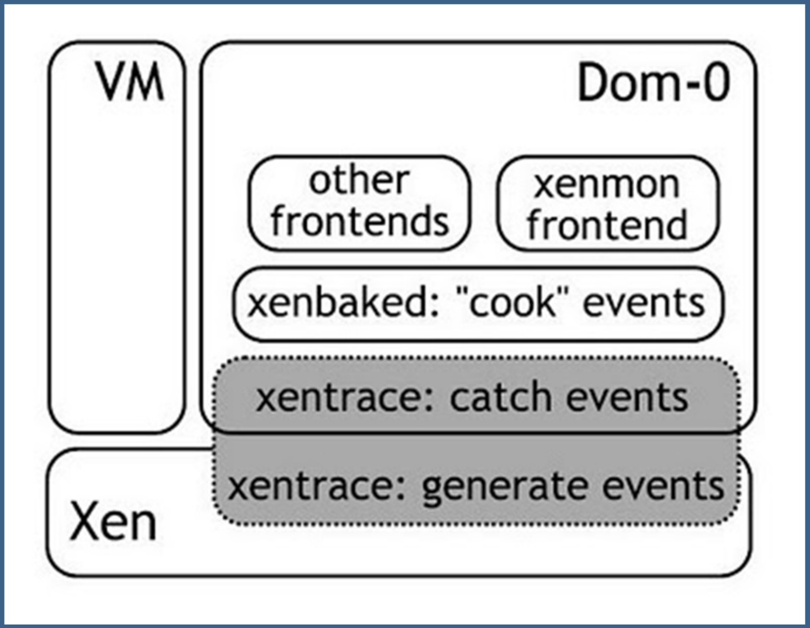
Information Gathering

- XenMon
  - CPU per VM per Core: usage, blocked, waited
- OProfile
  - L2 Cache hits and misses
  - Patch linux kernel
- DomU Kernel
  - MB/s read and write per VM

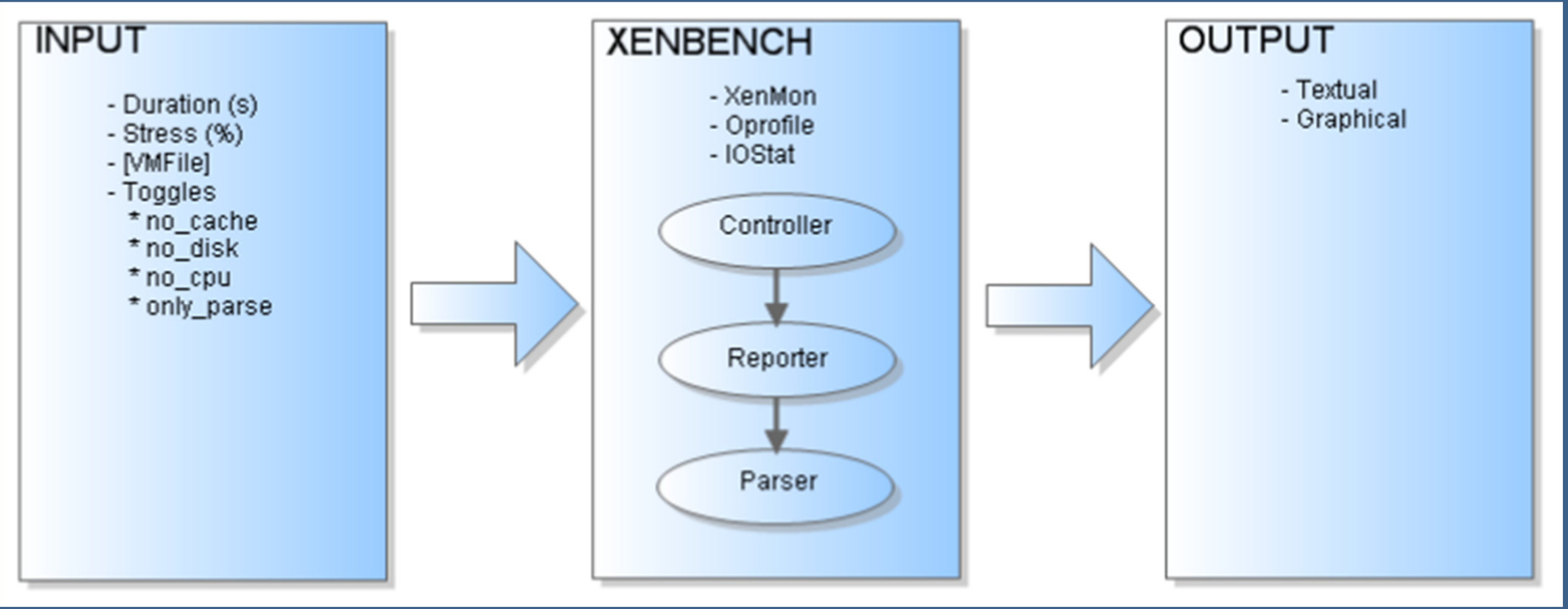
```
~$ iostat
```

Xenbaked

- Creates samples based on XenTrace events in XenBaked.
- Determine CPU usage per VM in frontend based on Shared Memory records (mmap).



Overview


INPUT
- Duration (s)
- Stress (%)
- [VMFile]
- Toggles
* no_cache
* no_disk
* no_cpu
* only_parse
XENBENCH
- XenMon
- Oprofile
- IOStat
Controller
Reporter
Parser
OUTPUT
- Textual
- Graphical

Output

===
CPU
===

VM	Core	0	1	2	3	4	5	6	7	total
0	cpu(%)	0.0	1.2	1.5	0.0	0.8	0.0	0.0	0.0	3.5
	blocked(%)	0.0	92.3	93.1	0.0	90.7	0.0	0.0	0.0	276.2
	waited(%)	0.0	0.2	0.2	0.0	0.1	0.0	0.0	0.0	0.5
1	cpu(%)	0.0	12.0	18.6	0.0	65.2	0.0	0.0	0.0	95.8
	blocked(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	waited(%)	0.0	0.1	0.1	0.0	0.1	0.0	0.0	0.0	0.3
2	cpu(%)	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	100.0
	blocked(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	waited(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	cpu(%)	100.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.3
	blocked(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	waited(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	cpu(%)	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	100.0
	blocked(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	waited(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	cpu(%)	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	100.0
	blocked(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	waited(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	cpu(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	100.0
	blocked(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	waited(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	cpu(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
	blocked(%)	0.0	19.7	55.6	0.0	23.9	0.0	0.0	0.0	99.3
	waited(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	cpu(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	blocked(%)	0.0	84.8	9.6	0.0	0.0	0.0	0.0	0.0	94.4
	waited(%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
total	cpu(%)	100.3	13.3	20.2	100.0	66.0	100.0	100.0	100.0	599.8
	blocked(%)	0.0	196.8	158.4	0.0	114.7	0.0	0.0	0.0	469.9
	waited(%)	0.0	0.3	0.3	0.0	0.3	0.0	0.0	0.0	0.9

Overhead

- Measure by using a CPU Benchmark
  - Not free: SPEC CPU2006, MultiBench,…
  - No multicore support: CoreMark, SuperPi,…
  - Solution: LinPack, determines the MFLOPS the system is able to achieve

- Comparing the amount of MFLOPS on an idle system and on a system running our tool, gives an idea about the overhead.

Conclusion

- What to do next?
  - Finish the CLI tool & create GUI
  - Try to further minimize measurement errors
  - Analyze and minimize overhead of the tool
  - Documentation of tool & online release

- Questions?

- Blog: http://www.stage.kurtvermeersch.com