Performance testing - why, how and what for?

A painful learning experience on simple slides

Marko Karg & Dustin Black
Red Hat Storage and Hyperconverged Architecture Team
WHY?
Comparability of different HW configurations

- Specs alone are insufficient:
  - Only per-subsystem: IOPS for storage devices, MHz for CPUs, throughput for network devices
- No way to deduce the expectable overall performance
  - No holistic information
- One subsystem may perform great on its own but extremely poor in conjunction with others
- Different workloads require a different balance of resources
Example same workload - Different HW Configs

It's always tempting to select the highest-performing results
Example same workload - Price-Performance

Design choice has a large impact on the efficiency of your $$
Example same workload - Impact of Design Choices
Comparability of same HW configurations

- Small changes - huge effects
- Typical examples:
  - RAID controllers: Caching policy writethrough vs writeback
  - Jumbo frames enabled / disabled on network devices
  - `vm.dirty_ratio`
  - `vm.dirty_background_ratio`
  - `directIO` vs. buffered IO
HOW?
Fundamentals

● We’re not testing a single system!
● Check all involved subsystems / Smoke tests:
  ○ 10GigE network - throughput?
    ■ iperf
  ○ HDD / SSD / NVMe - throughput / latencies?
    ■ fio / dd
  ○ CPU - tricky.
    ■ Fio cpu cycle burner, but that’s not telling us much
Fundamentals

- Repeat every test at least 3 times, better 5
- Calculate the standard deviation (sanity check)
  - Find surprises like caching effects
- Change one variable at a time!

Don't do this
Fundamentals - Monitoring

- Collect system submetrics of ALL involved systems
  - In a client - server setup, the clients might not be powerful enough to generate enough load
- Determine which system resource is the first to get saturated
  - Typically storage, but of course depends heavily on the test
- Make sure not to overdo as well
Fundamentals - How much do we monitor?

- Carefully choose what to monitor
- Easy to overdo, almost impossible to correlate
  - E.g. full monitoring in large numbers of short-lived VMs?
  - Load generators
- Spend time on meaningful metric combinations
Fundamentals - Metrics to monitor

● Gather important system information
  ○ CPU usage
  ○ Memory saturation / paging
  ○ Storage device load
  ○ Network throughput

● These are local metrics on a per host basis

● Complicated to correlate if multiple systems are in use
Fundamentals - Be careful what you're looking at

- Do you need total counts? Averages? Max or min? Percentiles?
- Merge distributed tests or measure per-client?
- Most metrics can be reported in any of these ways
- Example: Consider latencies
  - Monitoring averages may be deceiving
  - Max within a percentile may better represent acceptable user experience
Fundamentals - Local metrics

- Typical tools:
  - sar
  - vmstat
  - top
  - iotop

- Trending and correlation needs to be done manually
Fundamentals - Remote Metrics

- **collectd**
  - System daemons and modules to collect metrics
  - Metrics are pushed to a remote database

- **prometheus**
  - Uses metrics already reported by the kernel
  - Node exporter hosts data on a local port for the remote server to pull
  - A push gateway is available where pulling is not possible
Fundamentals - Data Visualization

- **Graphite**
  - Passive metrics collection via selection of receivers managed separately
  - Time-series graph rendering
  - Care must be taken to not lose data resolution over time
  - Not user-friendly using its own visualization

- **Prometheus**
  - Active metrics collection via native pull scraping
  - Time-series graph rendering with a more robust query language and native alerting
  - Graphs not really usable directly

- **Grafana**
  - Build highly customized and flexible dashboards of Graphite or Prometheus charts
Fundamentals - Workload-Tools

- **fio**
  - Flexible IO, versatile workload generator
  - Used to test HDD / SSD in the smoke tests
  - Throughput and latency tests on distributed file systems
  - Can do sequential or random access
  - Multi-host, synchronized tests
  - Highly customizable

Fundamentals - Workload-Tools

- smallfile
  - Multi-host tool to create small file workloads
  - Distributed tests can be synchronized
  - Extensible and scriptable
  - Incompressibility of generated data (important for some compressing SSD controllers)

Source: [https://github.com/distributed-system-analysis/smallfile](https://github.com/distributed-system-analysis/smallfile)
Fundamentals - Workload-Tools

- Iozone
  - Single / multiple stream measurement
  - Multi-process measurement
  - Various I/O types (posix, mmap, normal)

Source: [http://www.iozone.org/](http://www.iozone.org/)
Distributed client tests

- A single client probably can’t saturate a distributed system
- Testing tools need to be aware of their distribution:
  - Test starts and ends need to be in sync
  - Consider whether you need stonewalling
  - Load distribution needs to be taken care of
  - Results of all worker processes need to be centrally collected and analyzed
  - Most tools provide these functionalities
Distributed client testing

All clients need to start and stop the test in sync.

The control node starts the test on all nodes simultaneously.
Script your tools to make your life easier

- Each tool can be very flexible and complicated to configure
- You will want repeatability and a controlled set of variables
- It helps to have a more consistent user interface around the tools you use
- Consider using wrapper scripts or an existing overlay kit like pbench

Our wrapper scripts (disclaimer that these are rough around the edges):

https://github.com/red-hat-storage/storage-arch-tools
Workload-based: sysbench

- Benchmark suite to test
  - CPU
  - Memory
  - Threads
  - Mutex
  - File IO
  - Various databases
- Extensive statistics about rate and latency
- Extendable with scripts
Workload-based: SPEC SFS

- Highly-standardized benchmarking suite from SPEC
- Designed to test all components of a storage solution together as a whole
- Multiple synthetic workloads to simulate real-world usage scenarios
  - Database
  - Software build
  - Virtual desktop
  - Streaming video capture
  - Electronic design automation
- Protocol-independent -- Only requires a compatible client and a filesystem
- Strict reporting requirements to maintain test integrity
Full-stack testing:

- Makes use of all available types of system resource:
  - Network
  - CPU
  - Memory
  - Storage

- Typically consists of multiple, interacting components
  - Webserver
  - Database server
  - Application server
  - Client system(s)
Full-stack testing: DVD Store 2

- Simulates an online store that sells DVDs
- Customers login, browse and buy DVDs
- Number of customers is configurable
- Client - Database connection
- Mostly network / storage intensive
VM Density testing with DS2

RHHI-V Capacity Environment VM Density
Full-stack testing: DVD Store 3

- Similar to DVD Store 2
- Reviews and premium membership has been added
- Client - Webserver - Database
- Stresses CPU, Memory and IO more equally than DVD Store 2
DS3 latency testing

AVG OPM vs. STACKS

- AVG OPM
- avg latency per order (rt_tot_avg) in msec
- max latency (rt_tot_lastn_max) in msec
Automation in Full-Stack testing

- Ansible playbooks for:
  - Server VM Creation from a template
  - Client container instrumentation (target node, threads, db size)
  - VM tear-down
  - Cache clean-up
Questions?