Scalability: Pushing the Limits
PNSQC Presentation, October 2014

Neha Rai, Tim Schooley, Tejas Patil
So what is “Scalability”? 

“*Scalability* is the ability of a system to successfully handle an increasing workload, or its ability to be expanded without major architectural changes, or detriment.”

For a good read, check out “Characteristics of Scalability and Their Impact on Performance”, André B. Bondi, AT&T Labs
Once upon a time, there was...
Policy

Auditing

Reports

Key escrow

User authentication
Mission Critical

(Photographs for example only; not indicative of actual customers)
McAfee ePolicy Orchestrator

Drive Encryption

Agent-Server Communication Interval (ASCI)
Effects of changing the ASCI, with 100,000 clients

Average number of client requests per second vs. Agent-Server Communication Interval (hours)
So we integrated into ePO...

Key escrow

Policy

Auditing

Reports

Users
Are we ready to roll it out?

• Does it meet our scalability expectation?
  We had a number in mind, based on existing ePO scalability guidelines (goal of 100,000).

• Will it work for existing customers?
  Mission critical. It has to work.

• Does it meet our quality goals?
  Do we know what happens when the system reaches its limits?
Without testing the limits, bad things™ can happen.
Key take-away #1: Understand the risks of *not* doing Scalability Testing (this will help you determine if you need to do it)
What to test?

- Covers many components
- High impact failure case
- Simple result to interpret

- Covers high complexity code
- Covers a very common use case

"5_G>I’N^O!"

1.5x ASCI
OK, where are you going to get all the clients from?
(Note: this will depend on your architecture)

You might not have one of these!
“5_G>I’N^O!”

N nodes

N nodes

ePolicy Orchestrator

AH

DB

AH

N nodes

N nodes

"5_G>I’N^O!"
So why did we have to simulate?

(Optimization)

Not testing Steve’s true ability to cook under heavy demand.
So why did we have to simulate?

*Meaningful* data helps uncover the limitations of the system. *(for us, it was user data)*
Example causes of limitations

- Larger calculations
- Cache memory
- Connection pools
- Contention
- Disk IO
- Network IO

Recommendation: keep the hardware consistent, and don’t use virtualization unless you expect your customers to use it.
## General Info

- **Username:** CC4
- **New Token Timestamp:** 12941087137000
- **Scheduled Start Time:** 2011-02-02 02:26:17,000
- **Actual Change Time:** 2011-02-02 02:26:19,084
- **ePO Scheduled Task Delay:** 373 seconds

## Change Report

- **Updated Tokens:** 2282
- **First Updated Token Time:** 2011-02-02 02:32:32,576
- **Last Updated Token Time:** 2011-02-02 02:39:20,188
- **Update Rate (per second):** 0.00
- **Elapsed Propagation Time:** 6.8 minutes

## CVS Stats Dumping

- Enable CVS stats dumping

### Received token change for "CC54" with timestamp: 12941087138000

- 2011-02-02 02:42:57,122 : JU01_02254: Received token change for "CC54" with timestamp: 12941087138000
- 2011-02-02 02:42:57,122 : JU02_02110: Received token change for "CC27" with timestamp: 12941087139000
- 2011-02-02 02:42:57,122 : JU01_02363: Received token change for "CC54" with timestamp: 12941087138000
- 2011-02-02 02:42:57,137 : JU02_00015: Received token change for "CC24" with timestamp: 12941087138000
- 2011-02-02 02:42:57,137 : JU01_02150: Received token change for "CC54" with timestamp: 12941087138000
- 2011-02-02 02:42:57,137 : JU02_00184: Received token change for "CC24" with timestamp: 12941087138000
- 2011-02-02 02:42:57,137 : JU01_00910: Received token change for "CC54" with timestamp: 12941087138000
- 2011-02-02 02:42:57,153 : JU02_01800: Received token change for "CC72" with timestamp: 12941087138000
- 2011-02-02 02:42:57,153 : JU01_01506: Received token change for "CC54" with timestamp: 12941087138000
- 2011-02-02 02:42:57,153 : JU02_00861: Received token change for "CC37" with timestamp: 12941087138000

### Server Busy Total

- JU01: 3
- JU02: 4

**Server Busy Total:** 7  
**Server Busy Count (Last hour):** 0  
**Server Busy Count (Last 5 minutes):** 0
Key take-away #2:

Define your test scenarios sensibly.

Suitable tools for gathering results

Keep acceptance criteria simple

Target complex areas

Aim for broad coverage
So how did we run the tests?

(the goal was 100k, but we needed to find the limit)
What were our findings?
(bearing in mind this was a new integration)

• The first scalability tests were *fireworks*.
  – Crashes, memory leaks, deadlocks.
  – All uncovering high severity defects.

• We identified bottlenecks, then optimized.
  – Expensive calculations.
  – Expensive SQL transactions.

• We finally obtained a level of confidence.
  – *Now* we’re ready to sell it.
The results

ePO, Agent Handler and SQL server hardware:
Dell PowerEdge R515, 2.6GHZ 6C, 8GB, 7.2K SATA
Dell PowerEdge R715, 2x 2.0GHZ 8C, 8GB, 15K SAS

ASCI: 4 hours
Nodes: 100,000
Average requests per second (to DB): ~7

All tests passed on this configuration.

Notes: no other point products were installed.
These results are advisory only.
How might this apply elsewhere?

So like, hbo go isn't working and I missed GoT and if it doesn't start working I'm gonna punch a [redacted] in the face.
Cost vs Gain

Law of diminishing returns

Investment ($ in pushing the limits vs [Confidence in] ability to meet demand
Key take-away #3:

Invest in Scalability appropriately (it’s a bottomless pit, if you want it to be)
Summary

• Understand the risks of your system not meeting its Scalability requirements.

• Define your test scenarios sensibly.

• Invest appropriately in Scalability testing.

• Have fun, and enjoy the fireworks!
Questions?
Neha_Rai@McAfee.com
Tim_Schooley@McAfee.com
Tejas_Patil@McAfee.com
Remember to take the in-app Presentation Survey!