IBM Security

BigFix Capacity Planning

Mark Leitch
BigFix Performance & Secure Engineering

May 15, 2018

Agenda

1. Capacity planning overview:
   Two words that will make you a capacity planning expert!

2. Components:
   Understanding the building blocks of capacity planning.

3. Getting BigFixy:
   BigFix specific approaches.

Capacity Planning Overview

Capacity Planning Overview

Capacity Planning Overview

The BigFix capacity planning guide is the official reference for everything we will be discussing today.

http://www-01.ibm.com/support/docview.wss?uid=swg27048326

“Capacity planning involves the specification of the various components of an installation to meet customer requirements, often with growth or timeline considerations.”

It has a lot of information. We will simplify things for you today.

Capacity Planning Overview

Two words that will make you a capacity planning expert!

It depends!

It depends primarily on:

- Infrastructure.
  CPUs, vCPU, network, commitment policies, network shaping, …
  Infrastructure can change quickly!
- Client workloads.
  Workloads, customization, minimum vs median vs peak, …

Unfortunately, nobody wants to hear “it depends” so we try to be as “general and as specific as possible”.

Capacity Planning Overview

There can be diverse goals for capacity planning within an organization:

- Application team: Typically wants great performance at reasonable cost.
- Virtualization team: Typically wants over-commitment with ease of deployment and recovery.
- Storage team: Typically wants central storage, RAIDed, with fiber or iSCSI (!) interfaces.
- Network team: Want you to use as little bandwidth as possible.

These goals can be contradictory.

- Responsibility typically falls to the application team to drive improvements.
- In order to do this, clear problems with cost effective solutions must be demonstrated.
- We will show you how to do this!

Components

Components

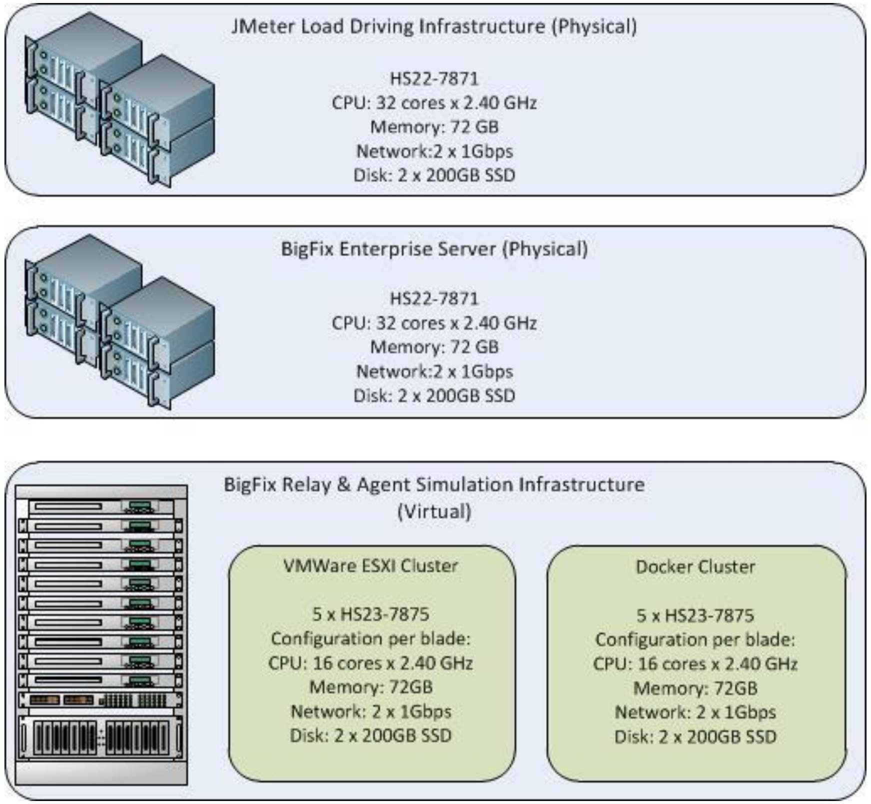
Components

Our capacity planning is based on internal benchmarks and field data, with the expected components of network, memory, CPU, storage.

- Network + memory are pretty easy.
- CPU is manageable.
- Storage is hard.

Core Philosophy: Ensure BigFix can run very well on “commodity hardware” (including virtual).

Performance Philosophy: Ensure as you throw hardware at BigFix, it simply goes faster. Not all sw does this!



Components: CPU

Moore's Law: The number of transistors in a dense integrated circuit doubles approximately every two years.

Everyone has become used to the continuous improvements in CPU.

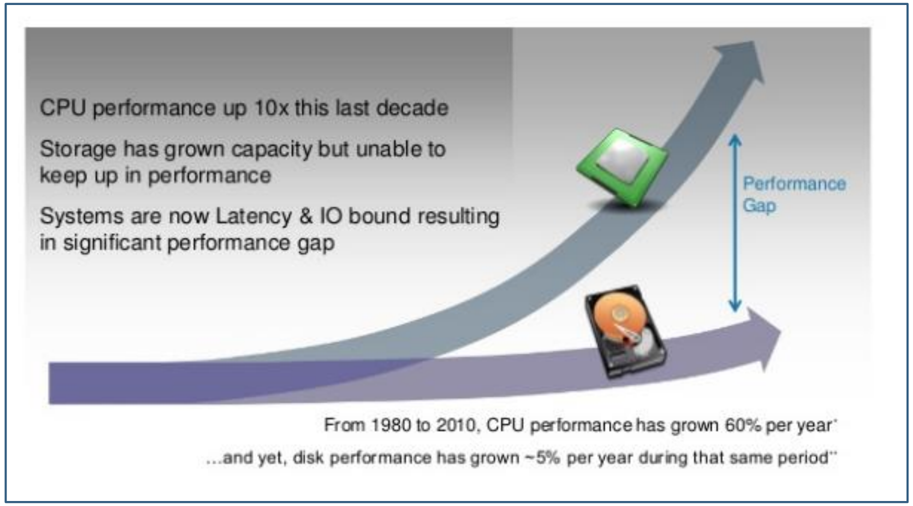
- Moore's Law is slowing down, and some may argue it is dead.

In terms of capacity planning, you can argue Moore's Law is dead.

- Cloud capacity planning is typically based on a 2.0+ GHz/core implementation, though 3.x GHz.core processors are relatively common.
    - Always take clock speed improvements if available.
- The cloud hypervisor will "steal" cycles. 90% efficiency is a good number.
- Hyper threaded cores represent about ≈30% of a true core.
- Meltdown/Spectre fixes alter the pipeline, and incur further degradation.

BigFix capacity planning is <u>simple</u> as we use the vCPU/commodity base.
Easy to meet the base, and go faster if you want to.
You <u>do</u> have to make sure your CPUs are happy. We will show you how.

Components: Storage



Storage has been the toughest area of performance for decades.

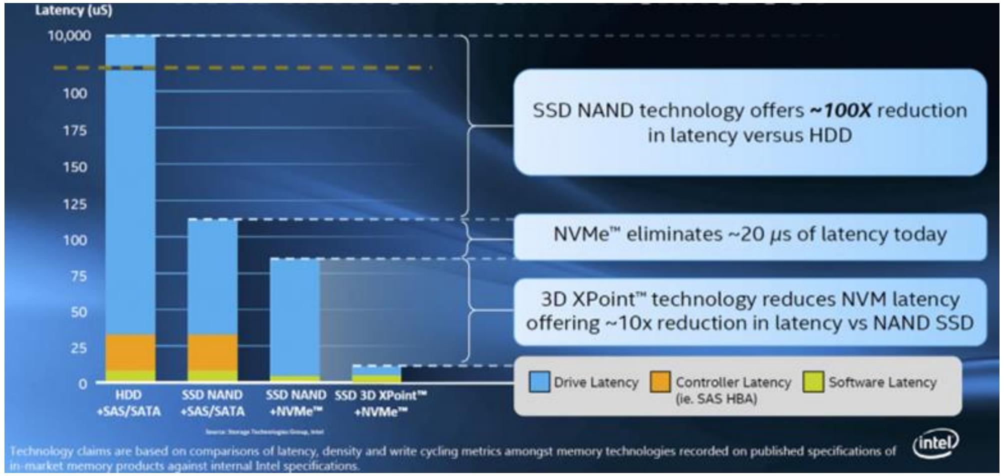
- Tapes, spinning platters, head assemblies, crashes, etc. Ugly stuff.
- Has lagged CPU improvements by a factor of 12x through the early 2000's.

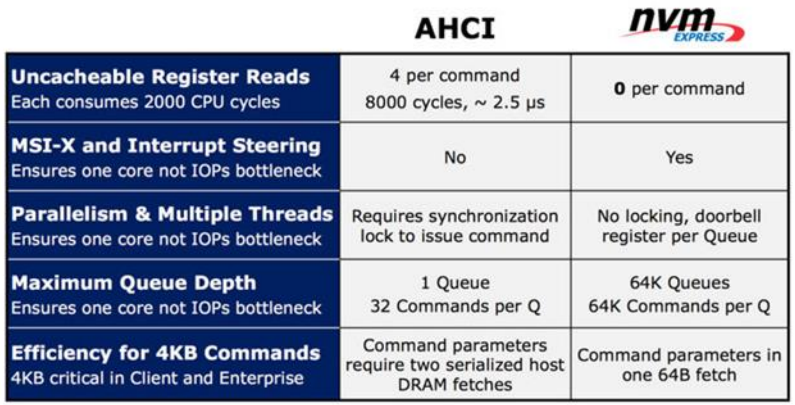
Flash has closed the gap.

- From 2000 → 2010, a 100x improvement.
- From 2010 → 2020, an aggregate 10,000x improvement.

But wait, it gets better!

Components: Storage





	AHCI	nvm EXPRESS
Uncacheable Register Reads Each consumes 2000 CPU cycles	4 per command 8000 cycles, ~ 2.5 µs	**0** per command
MSI-X and Interrupt Steering Ensures one core not IOPs bottleneck	No	Yes
Parallelism & Multiple Threads Ensures one core not IOPs bottleneck	Requires synchronization lock to issue command	No locking, doorbell register per Queue
Maximum Queue Depth Ensures one core not IOPs bottleneck	1 Queue 32 Commands per Q	64K Queues 64K Commands per Q
Efficiency for 4KB Commands 4KB critical in Client and Enterprise	Command parameters require two serialized host DRAM fetches	Command parameters in one 64B fetch

Source: Intel

The storage interfaces have changed dramatically.

- Massive reduction in drive, controller, and software latency.

Latency is the most important storage criteria for BigFix.

- Base requirement: 5,000+ IOPS with 1ms latency.

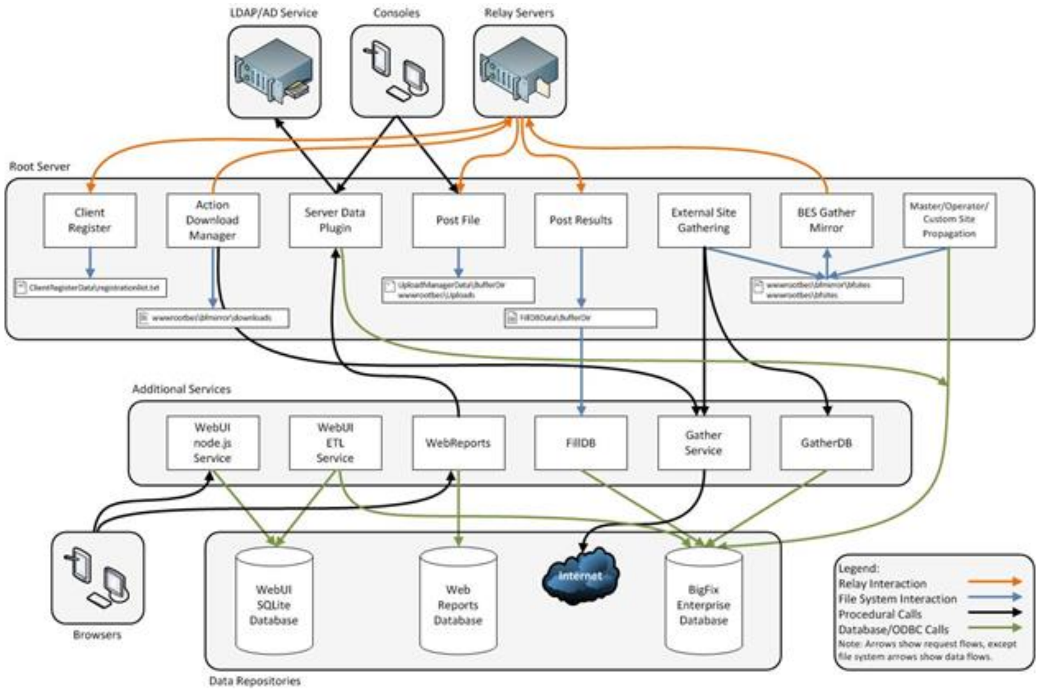
The number one field issue for capacity planning is high latency storage.

- We will show you how to measure and understand this!

Getting BigFixy

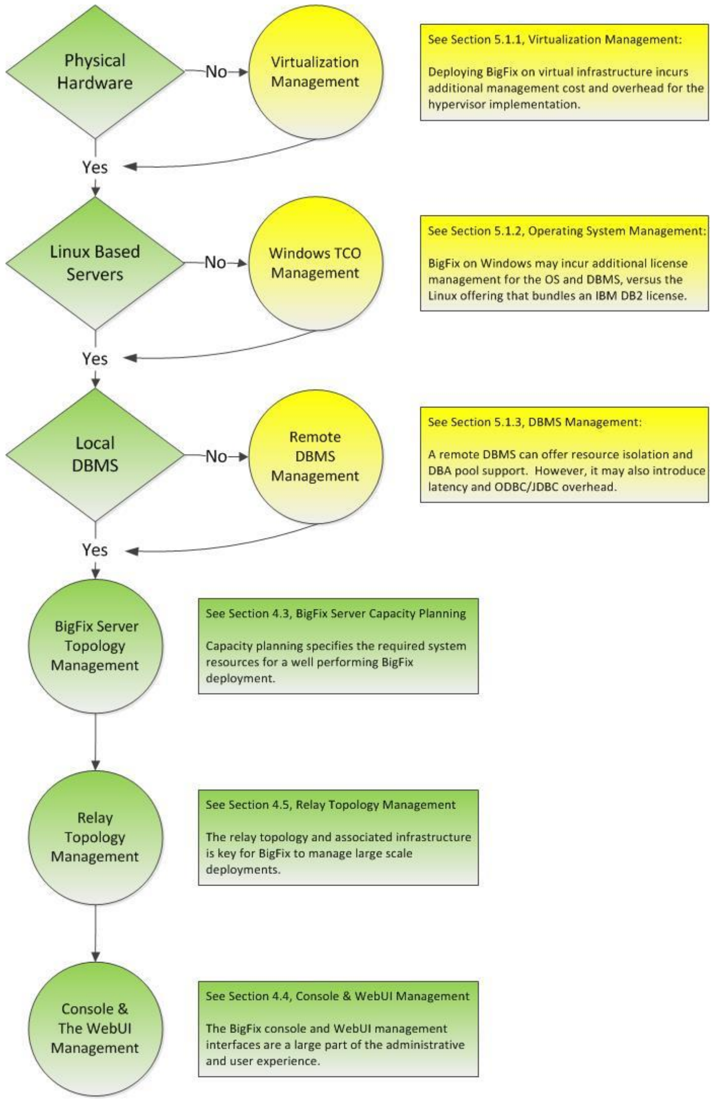
Getting Bigfixy

Getting BigFixy



BigFix has a large parallel architecture.

- We simplify capacity planning with a decision tree.


Physical Hardware
No
Virtualization Management
Yes
Linux Based Servers
No
Windows TCO Management
Yes
Local DBMS
No
Remote DBMS Management
Yes
BigFix Server Topology Management
Relay Topology Management
Console & The WebUI Management

See Section 5.1.1, Virtualization Management:

Deploying BigFix on virtual infrastructure incurs additional management cost and overhead for the hypervisor implementation.

See Section 5.1.2, Operating System Management:

BigFix on Windows may incur additional license management for the OS and DBMS, versus the Linux offering that bundles an IBM DB2 license.

See Section 5.1.3, DBMS Management:

A remote DBMS can offer resource isolation and DBA pool support. However, it may also introduce latency and ODBC/JDBC overhead.

See Section 4.3, BigFix Server Capacity Planning

Capacity planning specifies the required system resources for a well performing BigFix deployment.

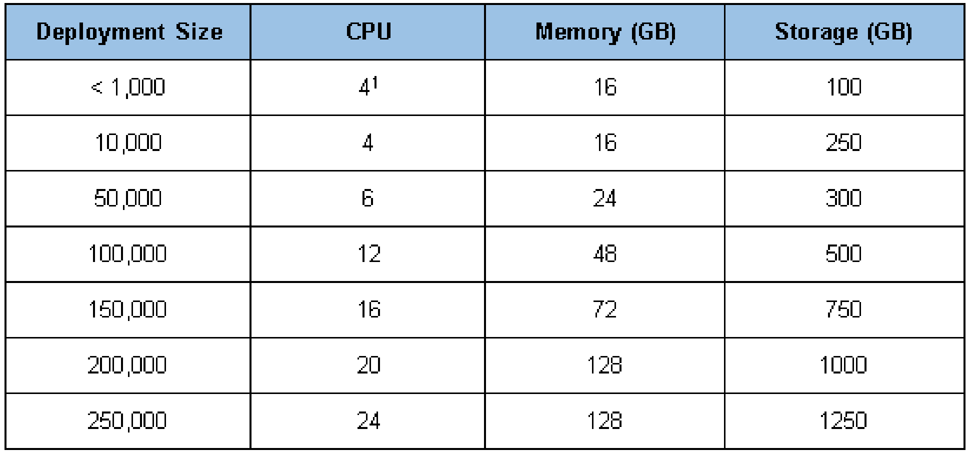
See Section 4.5, Relay Topology Management

The relay topology and associated infrastructure is key for BigFix to manage large scale deployments.

See Section 4.4, Console & WebUI Management

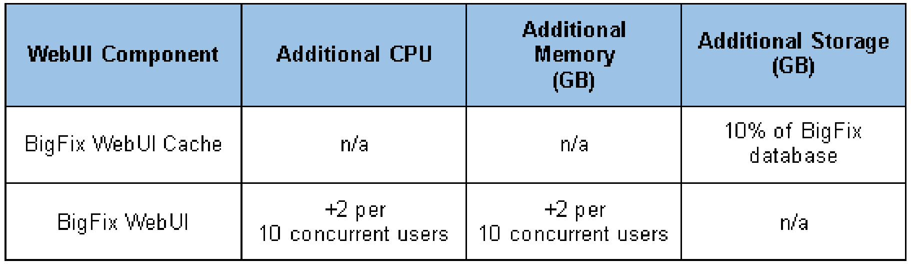
The BigFix console and WebUI management interfaces are a large part of the administrative and user experience.

Getting Bigfixy



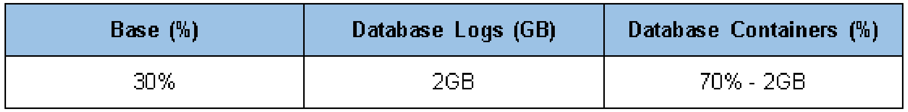
Deployment Size	CPU	Memory (GB)	Storage (GB)
< 1,000	4[1]	16	100
10,000	4	16	250
50,000	6	24	300
100,000	12	48	500
150,000	16	72	750
200,000	20	128	1000
250,000	24	128	1250

Figure 20: BigFix Management Server Capacity Planning Requirements



WebUI Component	Additional CPU	Additional Memory (GB)	Additional Storage (GB)
BigFix WebUI Cache	n/a	n/a	10% of BigFix database
BigFix WebUI	+2 per 10 concurrent users	+2 per 10 concurrent users	n/a

Figure 24: BigFix WebUIv2 Capacity Planning Requirements

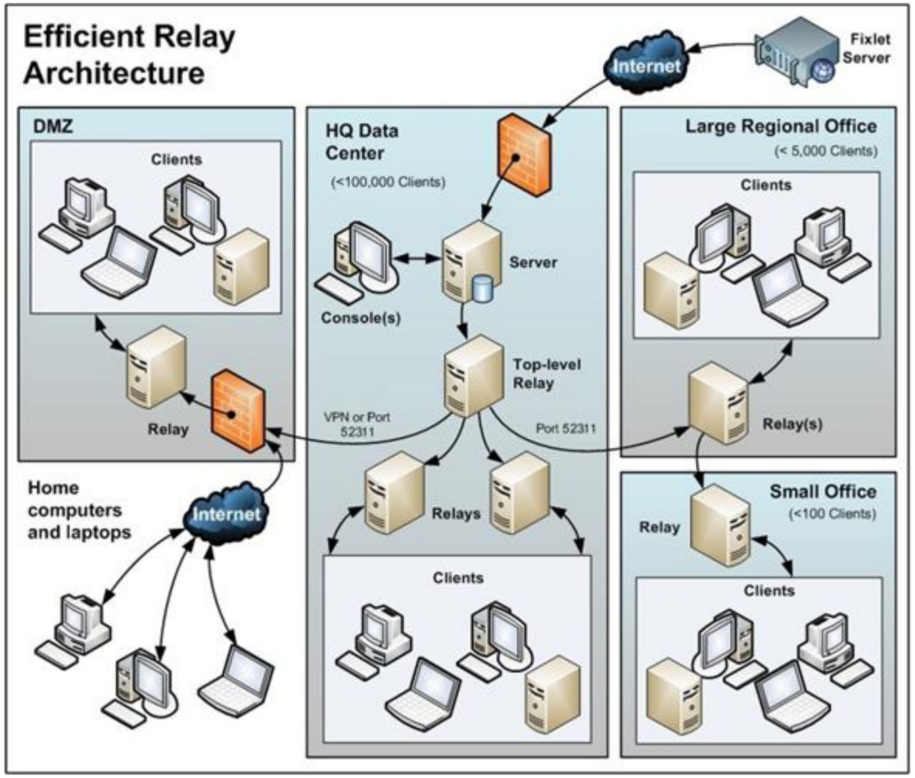


Base (%)	Database Logs (GB)	Database Containers (%)
30%	2GB	70% - 2GB

Figure 21: Storage Requirements Breakdown

- There is no “best” configuration, every organization has priorities: virtualization, management cost, DBA support, HADR, etc.
- Base model: Collocated root server and DBMS with remote WebUI, Web Reports server etc.
- CPU/memory/storage/network breakdown via the capacity planning guide.

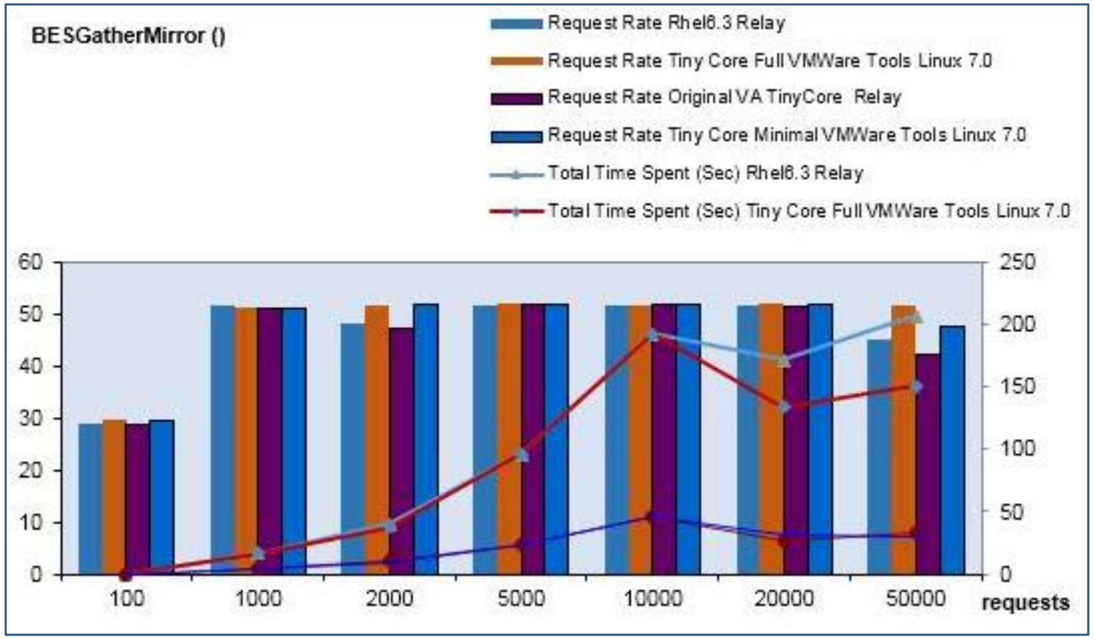
Getting Bigfixy



Relay capacity planning is simple ratio management:

- Leaf Relays: 1: 1000 endpoints (base) or 1:5000 (high scale).
- Top Level Relays: 1:120 relays.
- Top level relays are generally recommended once you approach 40,000 endpoints or over 100 relays (whichever comes first).

Getting Bigfixy



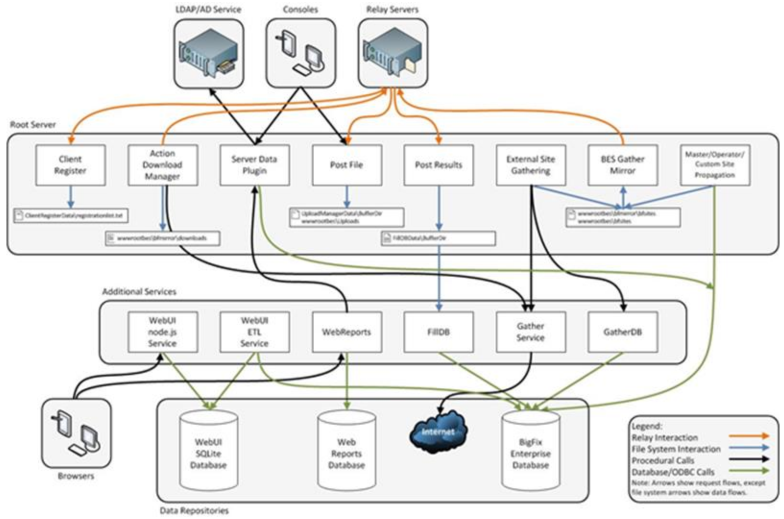
Relays virtualize extremely well.

- Tiny Core distribution (20MB distribution that can run in 64MB) offers beautiful results under load.
- We can run with 1 vCPU, but I always recommend a minimum of 2. Rationale: Cheap and by default more throughput.

Getting Bigfixy

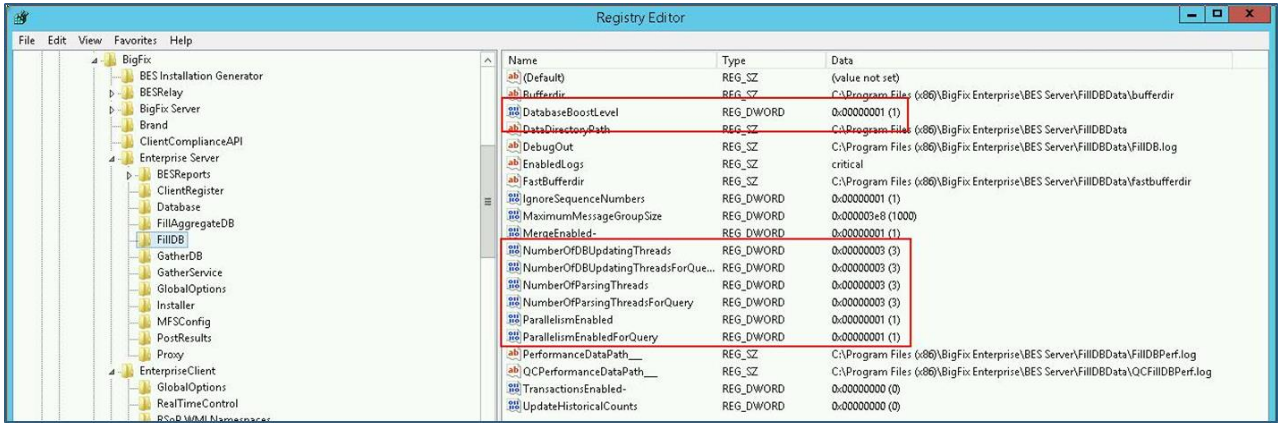
The BigFix FillDB service is responsible for processing endpoint reports and is critical for management at scale. In terms of processing it iterates over the following steps.

- Reads the buffer directory content.
- Parses the report, which includes:
  - Decrypting the encrypted reports.
  - Decompressing the compressed reports.
- Storing the report data in database tables.
- Replicating content from other DSA servers (optional).



The health of FillDB is a very good indicator of capacity planning and overall system health.
e.g. the ability to process the incoming buffer directory and drive high database insert/update rates (varies by target table type).

Getting Bigfixy



```
BigFix
    BES Installation Generator
    BESRelay
    BigFix Server
    Brand
    ClientComplianceAPI
    Enterprise Server
        BESReports
        ClientRegister
        Database
        FillAggregateDB
        FillDB
        GatherDB
        GatherService
        GlobalOptions
        Installer
        MFSConfig
        PostResults
        Proxy
    EnterpriseClient
        GlobalOptions
        RealTimeControl
        RSoP WMI Namespaces
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
Bufferdir	REG_SZ	C:\Program Files (x86)\BigFix Enterprise\BES Server\FillDBData\bufferdir
DatabaseBoostLevel	REG_DWORD	0x00000001 (1)
DataDirectoryPath	REG_SZ	C:\Program Files (x86)\BigFix Enterprise\BES Server\FillDBData
DebugOut	REG_SZ	C:\Program Files (x86)\BigFix Enterprise\BES Server\FillDBData\FillDB.log
EnabledLogs	REG_SZ	critical
FastBufferdir	REG_SZ	C:\Program Files (x86)\BigFix Enterprise\BES Server\FillDBData\fastbufferdir
IgnoreSequenceNumbers	REG_DWORD	0x00000001 (1)
MaximumMessageGroupSize	REG_DWORD	0x000003e8 (1000)
MergeEnabled-	REG_DWORD	0x00000001 (1)
NumberOfDBUpdatingThreads	REG_DWORD	0x00000003 (3)
NumberOfDBUpdatingThreadsForQue...	REG_DWORD	0x00000003 (3)
NumberOfParsingThreads	REG_DWORD	0x00000003 (3)
NumberOfParsingThreadsForQuery	REG_DWORD	0x00000003 (3)
ParallelismEnabled	REG_DWORD	0x00000001 (1)
ParallelismEnabledForQuery	REG_DWORD	0x00000001 (1)
PerformanceDataPath__	REG_SZ	C:\Program Files (x86)\BigFix Enterprise\BES Server\FillDBData\FillDBPerf.log
QCPerformanceDataPath__	REG_SZ	C:\Program Files (x86)\BigFix Enterprise\BES Server\FillDBData\QCFillDBPerf.log
TransactionsEnabled-	REG_DWORD	0x00000000 (0)
UpdateHistoricalCounts	REG_DWORD	0x00000000 (0)

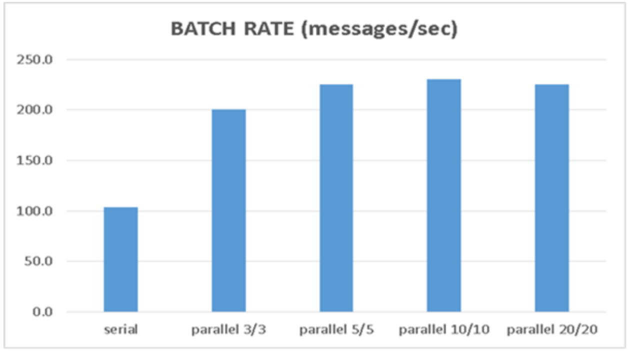
In BigFix 9.5.5 we introduced parallel FillDB to dramatically improve throughput and latency tolerance.

- Parallelism is enabled “out of the box”, with settings for both base reports and BigFix Query responses:
  Sample: Base report concurrency of [3 readers, 3 writers] if the machine has 6+ cores.
  Sample: Query report concurrency of [3, 3] also enabled if the machine has 10+ cores.
- Complete set of tuning options in the BigFix Knowledge Center (URL), but all “out of the box”.

We have been able to demonstrate a 240% performance improvement via a modest [3,3] thread configuration.

- Very nice adaptive scaling, with some CPU impact for additional parallelism.
  For example, with [3,3] threads we consume an additional core on Windows, and 1.5 cores on Linux.

Getting Bigfixy



Short + Long Batch Rate Normalized per Thread

How much faster can FillDB go? Let’s “cut to the chase”.
We have taken our already impressive improvements on FillDB and gone further. Much further.

- IOPS Sample Workload: > 100,000 IOPS with latency < 0.5 ms. This is 2000% times the BigFix requirement!
- FillDB Workload Scenario #1: A 300% improvement with the IBM Storwize V5000!
- FillDB Workload Scenario #2: A 700% improvement with the IBM FlashSystem 900!

These improvements are stacked on top of the 240% improvement already delivered for FillDB.
This represents a whopping cumulative 940% improvement for our reference scenario.
This is a testimony to the code efficiency of BigFix (ref. Amdahl’s Law).

Getting Bigfixy

This is all well and good, but isn’t there a better way to manage this?

https://ibm.box.com/s/ofohz4bj3qxbwdmq22s3kzwmpcmtf4xf
aka https://bit.ly/2ILKDJS

We have a capacity planning tool available.

- Limited distribution for now (password protected in Box).
- A PDF document with attached distributions for Linux and Windows.
    - Why password protected? Simply limiting the distribution for May, 2018. Consider yourself elite!
    - What is the password? Uhhh… seriously? Not here.
- Provided as part of the distribution for this class.
    - Open the document, navigate to the attachments, and download the distribution.
    - Unzip/Untar/Unwhatever and run.
    - Compiled Python.
- Let’s see how it simplifies capacity planning.

Getting Bigfixy

```
usage: MXCapacity [-h] [--endpoints ENDPOINTS] [--concusers CONCUSERS]
                  [--service {root,dbms,relays,webui,webreports} [{root,dbms,relays,webui,webreports} ...]]
                  [--platform {linux,windows}] [--relayscale {normal,high}]
                  [--mle] [--dump] [--format {table,csv,json}]

BigFix Capacity Planning

optional arguments:
  -h, --help            show this help message and exit
  --endpoints ENDPOINTS, -e ENDPOINTS
                        The number of endpoints to provide the sizing for
                        (default=10000).
  --concusers CONCUSERS, -c CONCUSERS
                        The number of *concurrent* users expected for user
                        interface services (default=10).
  --service {root,dbms,relays,webui,webreports} [{root,dbms,relays,webui,webreports} ...], -s {root,dbms,relays,webui,webreports}
[{root,dbms,relays,webui,webreports} ...]
                        The service to provide the capacity results for
                        (default=root).
  --platform {linux,windows}, -p {linux,windows}
                        The platform to provide the capacity results for
                        (default=environment where this program is running).
  --relayscale {normal,high}, -r {normal,high}
                        Deploy a normal scale or a high scale relay
                        (default=high).
  --mle, -m             Have the relay sizing account for Message Level
                        Encryption (MLE) (default=false).
  --dump, -d            Dump the capacity planning tables used by the utility.
                        This overrides all other options.
  --format {table,csv,json}, -f {table,csv,json}
                        The format to use to display the results
                        (default=table).
```

Options should be relatively straightforward but…
there is a super secret hidden option just for you!

Getting Bigfixy

```
Service           CPUs    Memory (GB)    Storage (GB)    Comments
----------------  ------  ------------   --------------  ----------------------------------------------------------------------------------------------------
Root Server       7       25             227             Sizing is for the root server only.  A separate database server is requested.
Database Server   13      51             527             Sizing is for the database server only.  Database log allocation should be 2GB.
WebUI             +2      +2             +53             Resources may be added to the root server or a dedicated server instance.
                                                         Note: Storage should be added to DBMS server for WebUI specific tables.
Leaf Relays       [2:4]   [4:8]          See comments.   Resources are [minimum:recommended] value pairs.
                                                         Storage in GB = OS + 3GB (BigFix binaries + logs) + 2GB (Default Cache) + Extended Cache…
                                                         For 120000 endpoints the recommendation is for 24 high scale relays.
Top Level Relays  [2:4]   [4:8]          See comments.   Resources are [minimum:recommended] value pairs.
                                                         Storage in GB = OS + 3GB (BigFix binaries + logs) + 2GB (Default Cache) + Extended Cache…
                                                         For 120000 endpoints the recommendation is for 1 false root + 3 additional top level relays.
```

Based on our documented capacity planning requirements, just simplified.

- Trying to balance simplification with the rich diversity of deployments.

What is the secret option?

- --env {virtual, sotp}
- Virtual is our default, and once again all sizing is based on vCPUs.
- sotp = State of the Practice, meaning what can be realized with a modern physical CPU.
- Assumes a 3+ Ghz clock speed and will reduce cost statements based upon this.
- Why is it hidden? We need to do further validation of our cost statements.

Try it out. Any complaints let me know. We will use the other provided tools in our next sessions.

Summary

Most deployments look good on paper.

- Internal teams will often tell you all is “good”.
  However, their definition of “good” may not match yours.

It is easy to know if something is wrong, but often hard to know what to fix.

- It is good to throw hw at the problem, but you have to know *what* hw.
- We often see CPU upgrades planned, when storage is the real problem.

Our number one rule for deploying a new BigFix server: benchmark IO.

- Once again, we do very well on commodity hw (e.g. SATA based SSD).
- Plug in a storage array with 100,000 IOPS and 0.5ms latency, we go faster.

Our next topic will be how to do a health check, identify poorly performing components, and understand how to build and deploy a BigFix server that will meet or exceed your enterprise needs.

IBM Security

THANK YOU

FOLLOW US ON:

- ibm.com/security
- securityintelligence.com
- xforce.ibmcloud.com
- @ibmsecurity
- youtube/user/ibmsecuritysolutions

IBM®