Architecture and Performance characteristics of a PostgreSQL implementation of the TPC-E and TPC-V workloads

Andrew Bond (Red Hat), Doug Johnson (InfoSizing), Greg Kopczynski (VMware), and H. Reza Taheri (VMware)

Agenda/Topics

- Virtualization and virtualization benchmarks
- Historical perspective of TPC-V
- TPC-V design considerations and characteristics
- TPC-V architecture
- End-to-end Reference Kit
- Prototyping results, and PostgreSQL characterization
- Benchmark roadmap

Virtualization and need for a database benchmark

- **Virtualizing servers allows:**
  - consolidation
    - Reduces both CapEx and OpEx
  - Migration of VMs (both storage and CPU/memory)
    - Enables live load balancing
    - Facilitates maintenance
  - High availability and fault tolerance
- **Cloud computing is powered by virtualized servers**
- **Databases VMs ~~*are*~~ *were* the last frontier for virtualization**
  - Initial hesitation to put enterprise databases on VMs
  - Followed by today's push towards virtualizing everything
- **We need a benchmark to model all this!**

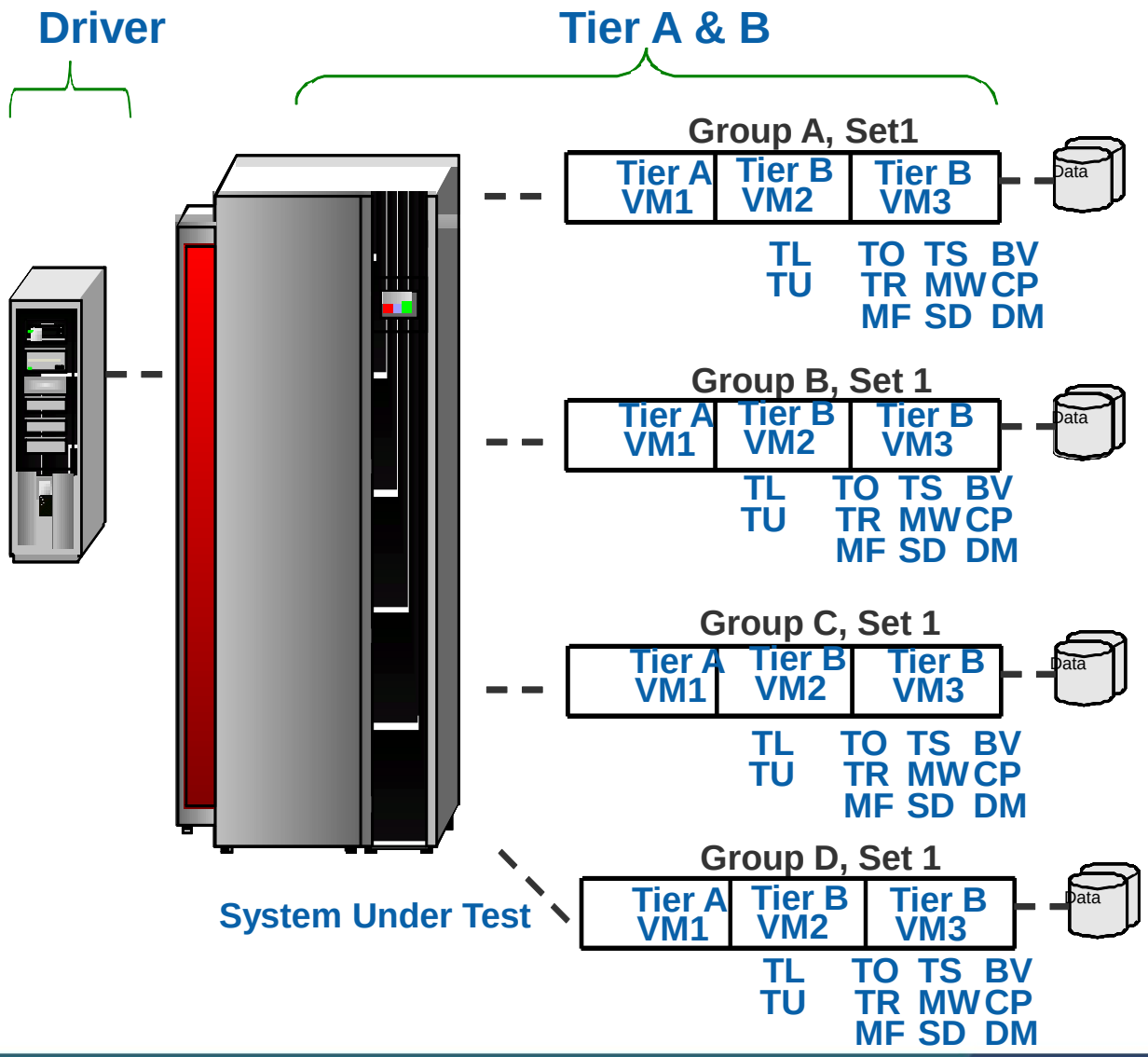
Today's virtualization benchmarks

- **VMmark**
  - Developed by VMware in 2007; now on version 2.5
  - De facto industry standard with nearly 200+ publications from 11 vendors
  - Released kit only runs on ESX, but can be modified for other hypervisors
  - Models ***consolidation*** of lightly loaded VMs with diverse workloads
- **SPECvirt_sc2010/SPECvirt_sc2013**
  - Industry standard, with 35 results from 4 vendors
  - Models ***consolidation*** of lightly loaded VMs with diverse workloads
- **TPC-VMS**
  - No results yet
  - Models ***consolidation*** of 3 identical database VMs
    - Test sponsors can use one of 4 existing TPC benchmark workloads
    - \- Limited in scope of what virtualization features get tested
    - \+ But does not require a ***new*** kit

History of TPC-V

- **2009 VLDB: We need a benchmark that:**
  - Models virtualized databases
  - Is industry standard
- **2010 VLDB: Proposal for a benchmark**
  - Steal TPC-E's workload, schema, specification
  - *Baby bear* # of VMs
  - Model elasticity of load
- **2012 VLDB: Status of the benchmark development**
  - Developing a **complete**, end-to-end , **publicly-available** kit
  - Kit runs on PostgreSQL
  - Finalized benchmark architecture
  - Some early result
- **This pace is lightening fast by TPC standards!**

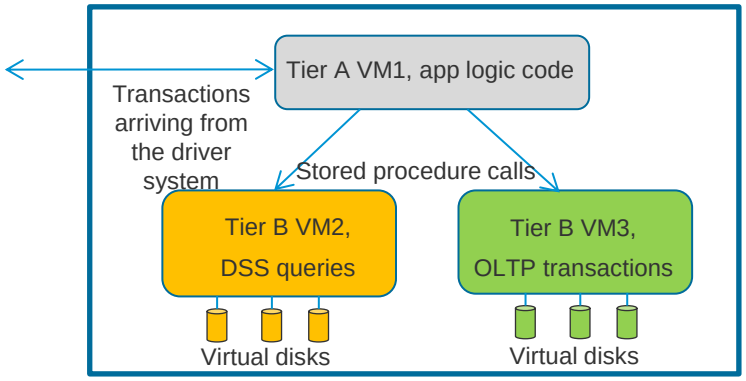
Components of a TPC-V configuration



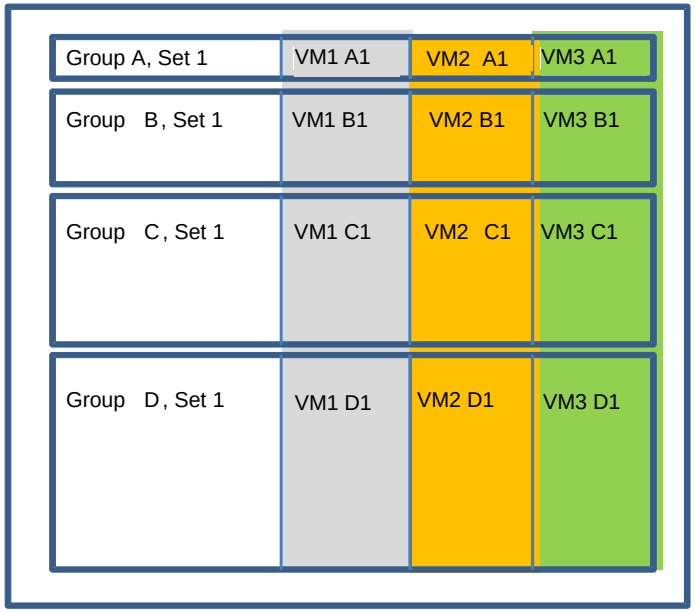
Sets and Groups

Group:	A	B	C	D
Average contribution to overall throughout	10%	20%	30%	40%

- **VMs with heterogeneous load levels**
  - Always 4 ***Groups*** @10%, 20%, 30%, 40%
- **But the number of Sets per Group grows with performance**
  - Small systems have 1 ***Set*** per Group
  - Larger servers divide **each** Group's load among 2, 3, … Sets
- **Total number of VMs on the server:**
  - 12 on small and medium-sized servers
  - 24 on today's high end
  - 36-48 on high end in 5-10 years



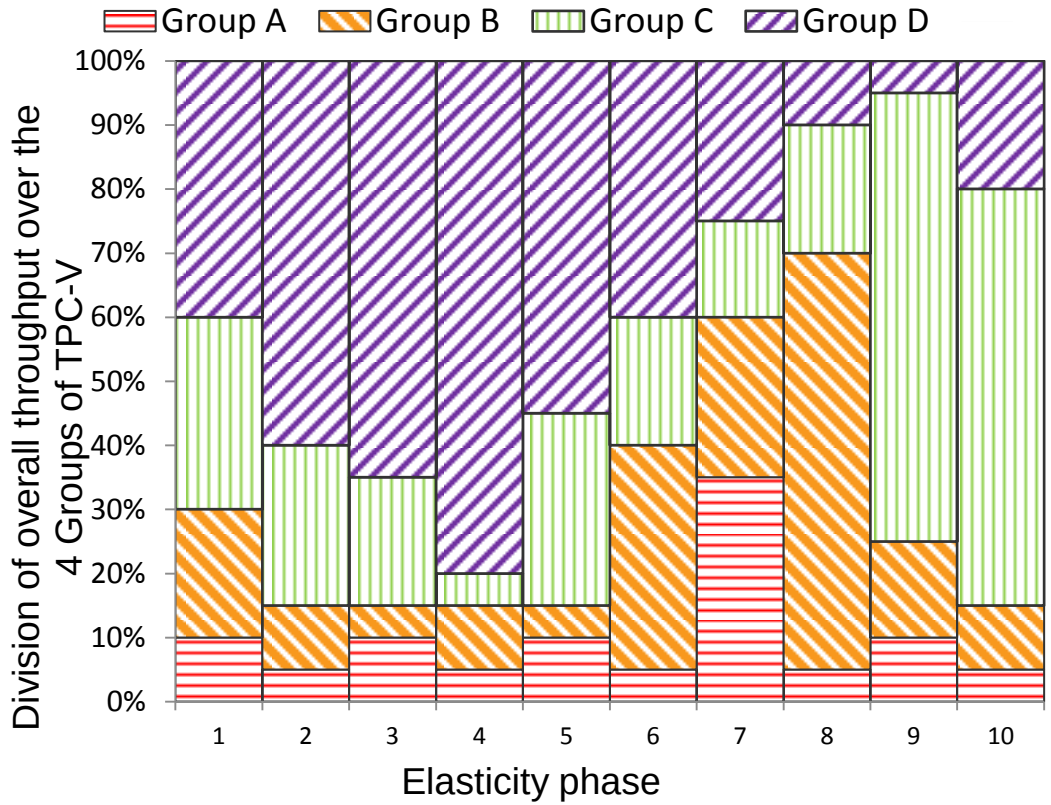
3 VMs of a TPC-V ***Set***



4 TPC-V ***Groups***, each with 1 Set

Elasticity

- **Load of each VM ranges by as much as 16X in ten 12-minute Phases**
  - Elasticity
  - Oversubscription

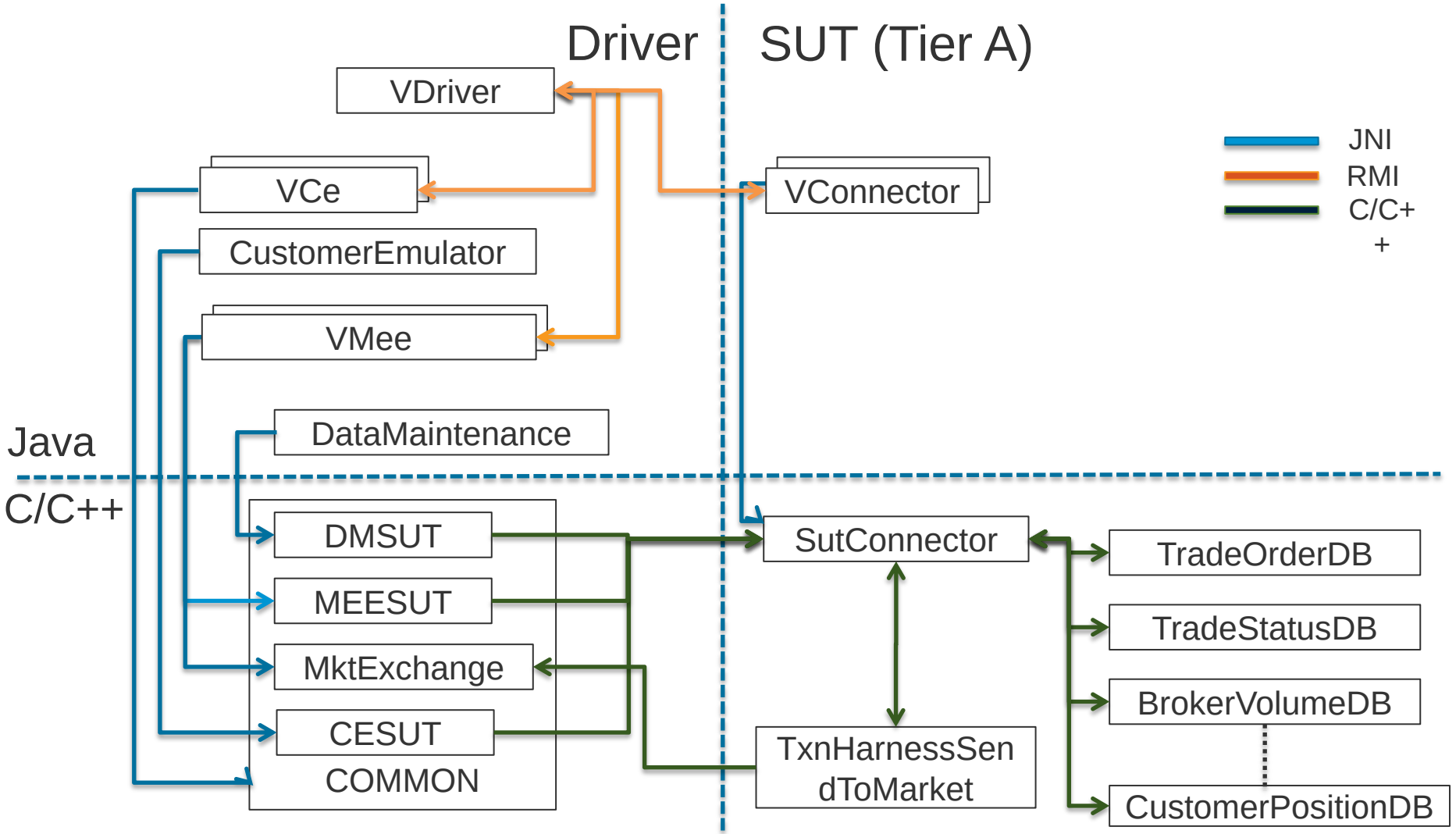


Period	Group A	Group B	Group B	Group D
1	10%	20%	30%	40%
2	5%	10%	25%	60%
3	10%	5%	20%	65%
4	5%	10%	5%	80%
5	10%	5%	30%	55%
6	5%	35%	20%	40%
7	35%	25%	15%	25%
8	5%	65%	20%	10%
9	10%	15%	70%	5%
10	5%	10%	65%	20%
Average	10%	20%	30%	40%

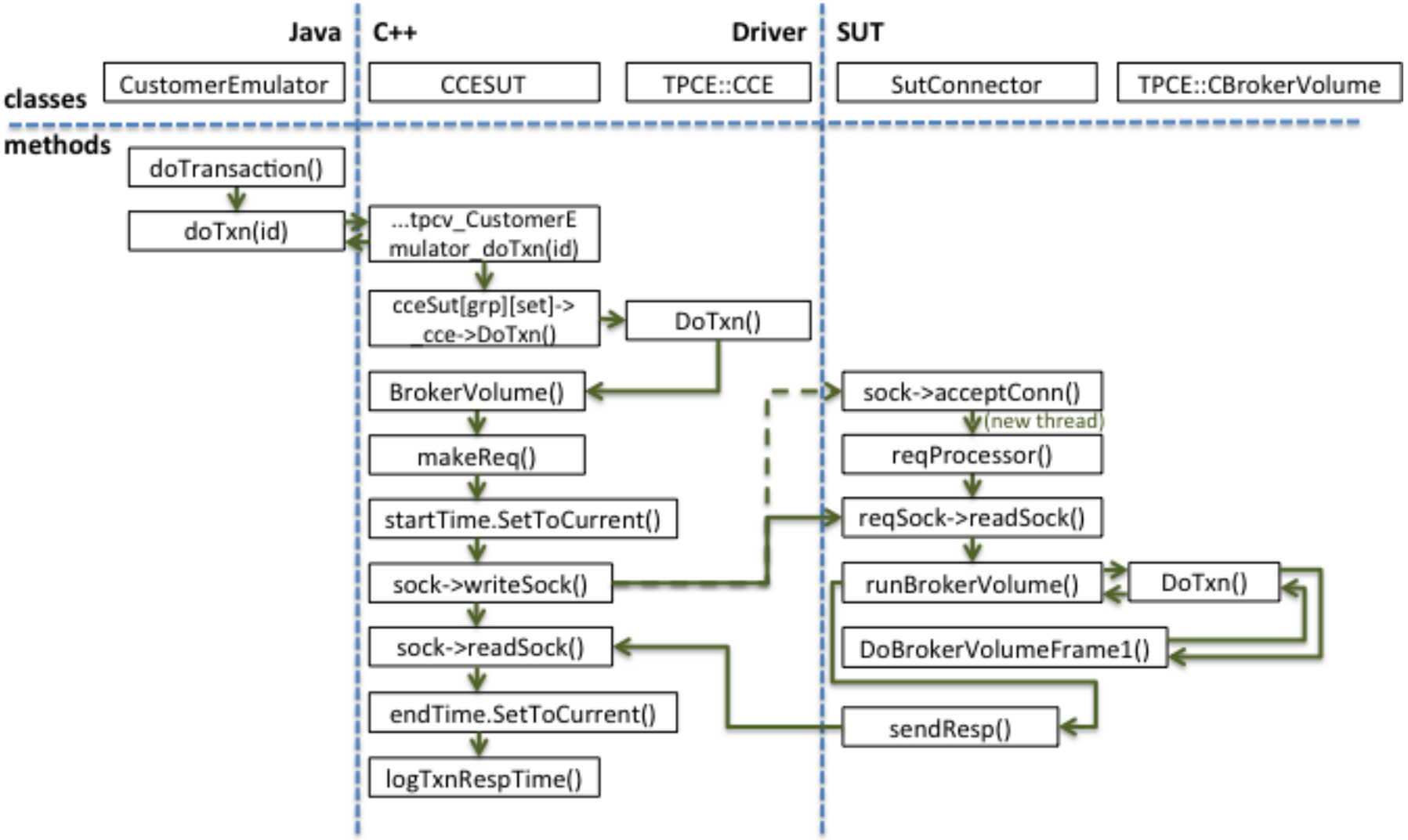
Design Considerations

- **Driver code in Java; transaction-specific code in C++ to match EGen/VGen**
- **Walk before you run: a TPC-E kit first**
- **ODBC allows easier database swapping**
  - Albeit with a performance cost
  - We can replace ODBC with native calls for best performance
- **Develop initial kit on PostgreSQL**
  - But ODBC makes it possible to run against other databases
- **Make the kit available to anyone, subject to a EULA**

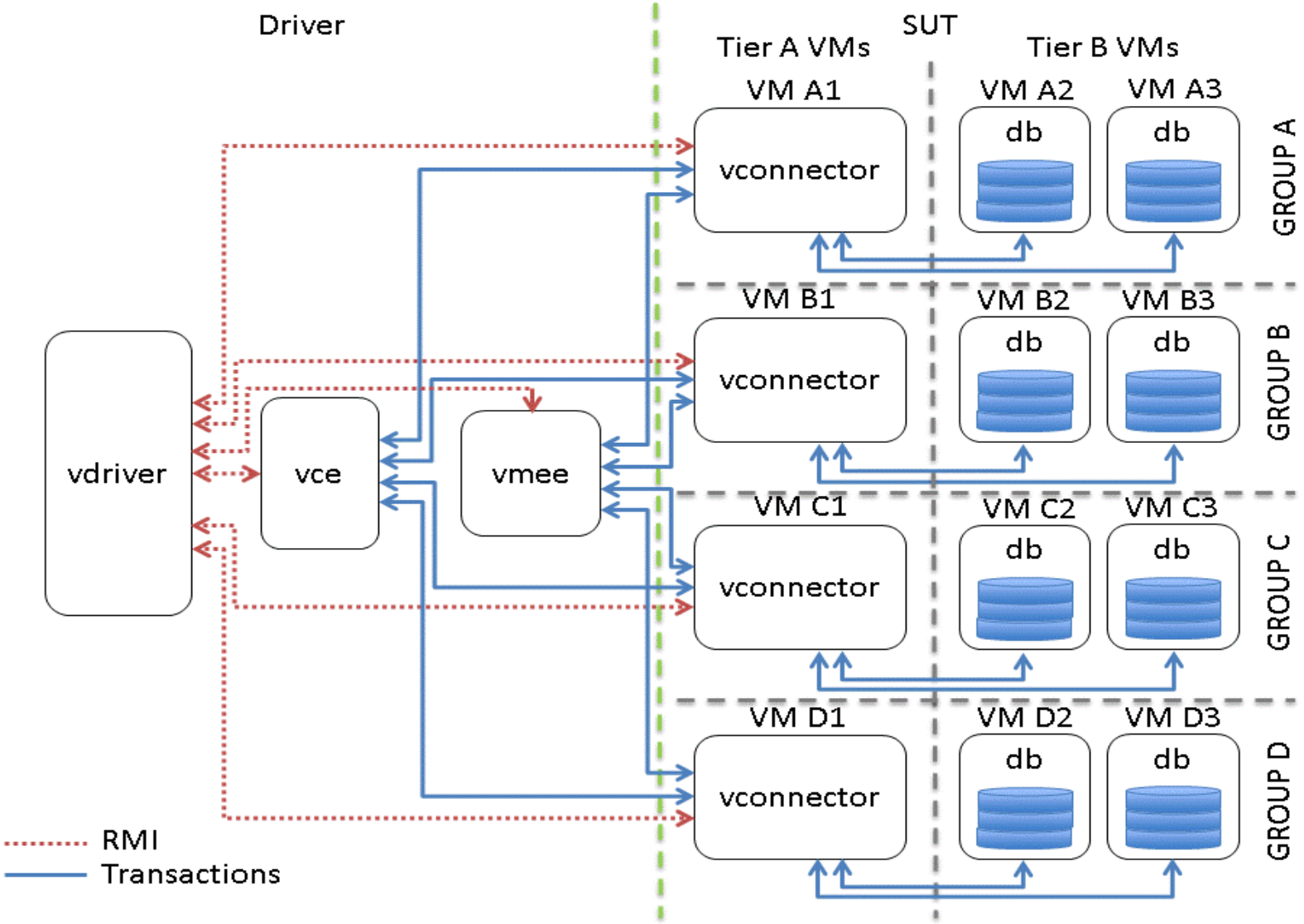
Java/C++ Class Interaction



Request Code Execution Path



Driver components



Status of the kit

- **We have a complete, end-to-end TPC-E kit**
  - This is TPC-E, so 1 database, static load
  - Scripts to create the schema and populate the database
  - All the necessary DDL/DML
  - C++ code to implement the business logic
  - Java driver to drive the load and collect and process the results
    - Handles multiple Sets and Groups, and elasticity
  - Linux shell scripts to start and monitor the run, and collect stats
- **The kit implements all the multi-VM and elasticity features of TPC-V**
  - Need to change the code for 2 transactions that are different in TPC-V
- **Busy prototyping!**

TPC-E results with a single database VM

- **Single VM/database (emulating TPC-E)**
- **4-socket HP ProLiant DL580 G7 server**
  - 2.40GHz Intel Xeon E7-4870 (WestmereEX) CPUs
    - So total of 40 cores/80 hyperthreads
  - 1TB of memory
  - Disclosed TPC-E result: **2,454 tpsE**
    - Other results on similar servers as high as 3,XXX tpsE
  - Two EMC VNX5700 disk arrays. 38 SSDs and 88 spinning disk drives
- **Running TPCE on a VM with 16-vCPU, 280GB Tier B VM**
  - So using 1/5th of the hardware resources
- **Software versions:**
  - VMware vSphere 5.1
  - RHEL 6.1
  - PGSQL 9.2.2
  - unixODBC 2.2.14

TPC-E prototyping results

- **Measuring 198 tpsE on the 16-vCPU VM**
  - Performance is decent for this stage of development
    - 1/13th of disclosed results on this server, but using 1/5th the resources
      - So our performance is 1/3 to 1/2 of commercial databases
    - 2 orders of magnitude higher than dbt5 experimental results
  - 85% CPU utilization
  - 19K IOPS, 212MB/sec
- **I/O rate appears to be our biggest problem**
  - High IOPS even though we have cut the DB size by 60%
  - IOPS/tran around 2X what commercial database does
  - PostgreSQL buffer management and file system relationship unknowns
  - Lack of clustered indexes
  - On-disk size appears as much as 20% larger

PostgreSQL tuning: File systems

- **This data is from when we separated the OLTP and DSS transactions into two different VMs**
- **Definitely should separate the log and data file systems**
  - So data blocks don't get flushed every time we write to the log
  - 6% more throughput
  - Lower response times
- **Use ext4 for data**
  - ext3 is fine for log

OLTP Trans		wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await
1 file system	Data +log	403	542	476	7682	5552	27	5.1	4.75
2 file systems	Data	194	860	145	15613	1357	34	6.3	6.29
	log	1	0.04	225	0.16	3066	27	0.3	1.15

DSS Trans		wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await
1 file system	Data +log	1830	11151	2767	138602	33956	25	30	2.14
2 file systems	data	2406	12350	2278	181902	18737	27	40	2.71
	log	343	0.34	134	1	17854	264	0.3	1.87

PostgreSQL tuning: File systems, continued

- **PostgreSQL folks like to encourage you to use the file system**
  - No! They insist that relying on the filsys buffer cache is the only way to go
- **Being an old database hand, you have to show me why this is so**
  - Double buffering wastes memory
  - Once you miss in the DBMS cache, you pay most of the price in OS cycle whether or not you hit in the buffer cache
- **I have been trying to max out PostgreSQL `shared_buffers`**
  - Ignoring the PostgreSQL book advice
  - Hard to shrink file system buffer cache when raising `shared_buffers`
    - End up swapping
    - I have to reboot the VM
- **But going from 34GB of `shared_buffers` to 200GB improves performance only 5%**
  - PostgreSQL/filsys interaction problem?
  - Or TPC-E benefits from DBMS cache reaching diminishing returns?

PostgreSQL tuning: checkpoints

- **Watch the number of 16MB `checkpoint_segments`**
  - On a high throughput system, this will decide checkpoint frequency
  - Default of only 3 `checkpoint_segments way too low`
  - Even with 128 `checkpoint_segments`, we were checkpointing every 2 minutes
  - Raised to 1,920 on the largest VM to checkpoints every 30 minutes

Checkpoint metric	128 segments	5,120 segments
checkpoints_timed	0	1
checkpoints_req	15	0
buffers_checkpoint	4,437,177	956,174
buffers_clean	14,069	852,893
buffers_backend	46,297	39,297
buffers_alloc	24,831,473	23,749,499

PostgreSQL tuning: Index-only scans

- **TPC-E transactions need clustered indexes**
  - All published TPC-E results have used MS SQL Server
  - They create clustered indexes on ALL 33 tables
  - (Secondary) indexes not used very often
- **PostgreSQL does not have clustered indexes**
  - Every data access has to read both the index block and the data block
  - Indexes are not much smaller than tables
  - Plus, PostgreSQL on-disk footprint is larger
- **Switched to Index-only scans with PostgreSQL 9**
  - But works only if all the columns named in the query are in the index
  - So have to create many, multi-column indexes
  - Index size grows quickly
    - After a week of runs, TRADE data blocks grew 4%, index blocks 40%

PostgreSQL tuning: Index-only scans, continued

Table	MS SQL			PostgreSQL		
	Table size	Index size	# of indexes	Table size	Index size	# of indexes
CASH_TRANSACTION	125GB	0.45GB	1	146GB	140GB	1
DAILY_MARKET	8.7GB	3.6GB	1	11GB	4.7GB	1
HOLDING_HISTORY	67GB	35GB	2	93GB	125GB	2
NEWS_ITEM	21GB	0.0003GB	1	20GB	0.003GB	1
SETTLEMENT	68GB	0.3GB	1	91GB	78GB	1
TRADE	153GB	82GB	3	176GB	135GB	3
TRADE_HISTORY	96GB	0.25GB	1	168GB	124GB	1

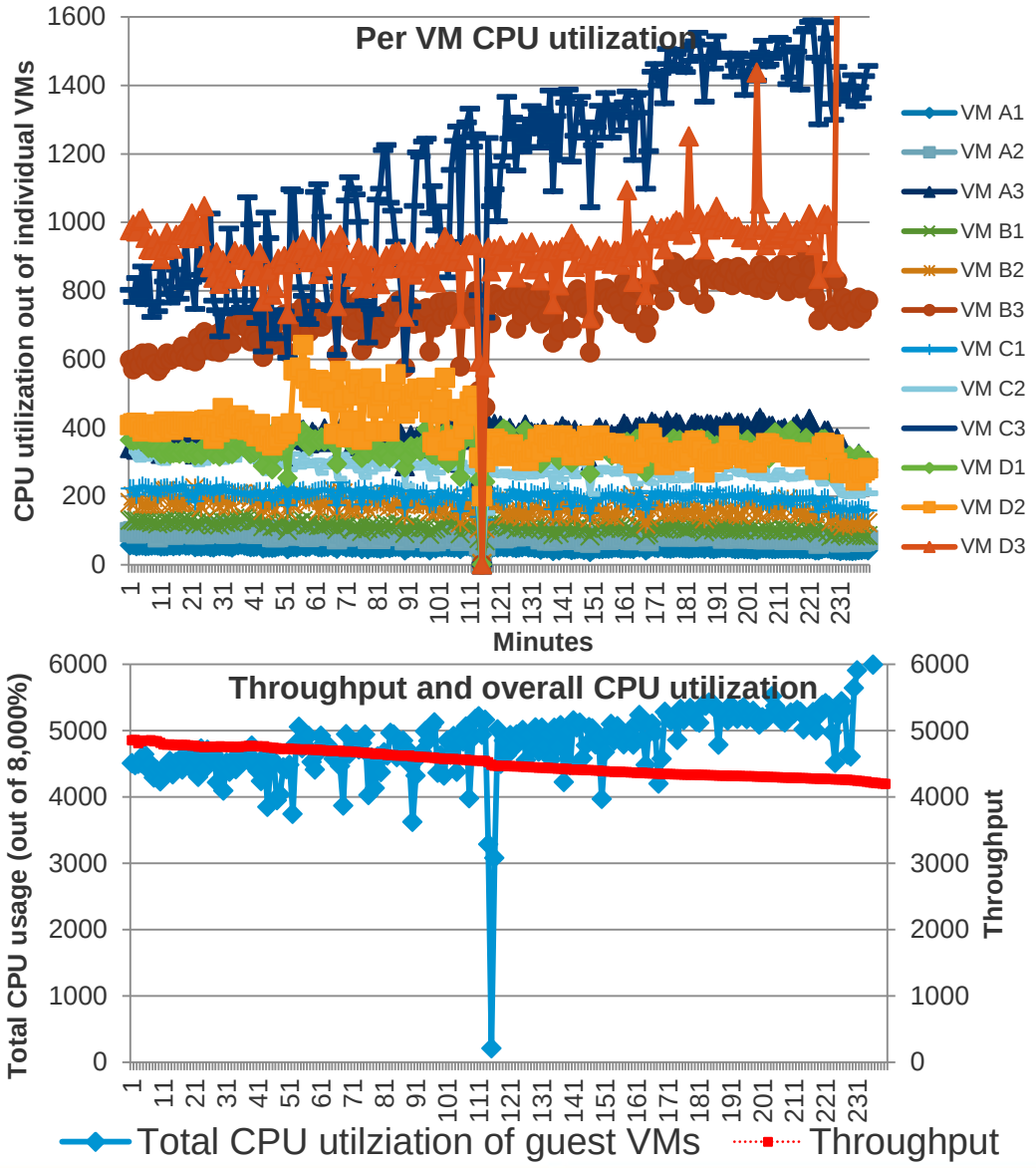
- **Note the index size for TRADE_HISTORY**
  - Even having Index-only scans doesn't reduce how much data we juggle

Server configuration for TPC-V prototyping

- **Carve out the 80-thread, 1TB server into:**
  - 4 Groups, each with 1 Set of 3 VMs
  - 12 VMs
    - 4 Tier A VMs (VM1s of Groups A-D) with 2GB of memory and 3-8 vCPUs
      - These Tier A VMs have low resource demands
    - 4 DSS Tier B VMs (VM2s of Groups A-D) with 88-278GB of memory and 4-16 vCPUs
      - High I/O load (hence more memory to cache more of the database)
    - 4 OLTP Tier B VMs (VM3s of Groups A-D) with GB of 39-78GB of memory and 12-40 vCPUs
      - Low I/O load but high CPU demands
  - Overall CPU allocation is overcommitted by 2X (168 vCPUs)
  - No memory overcommit
    - Not recommended for database VMs

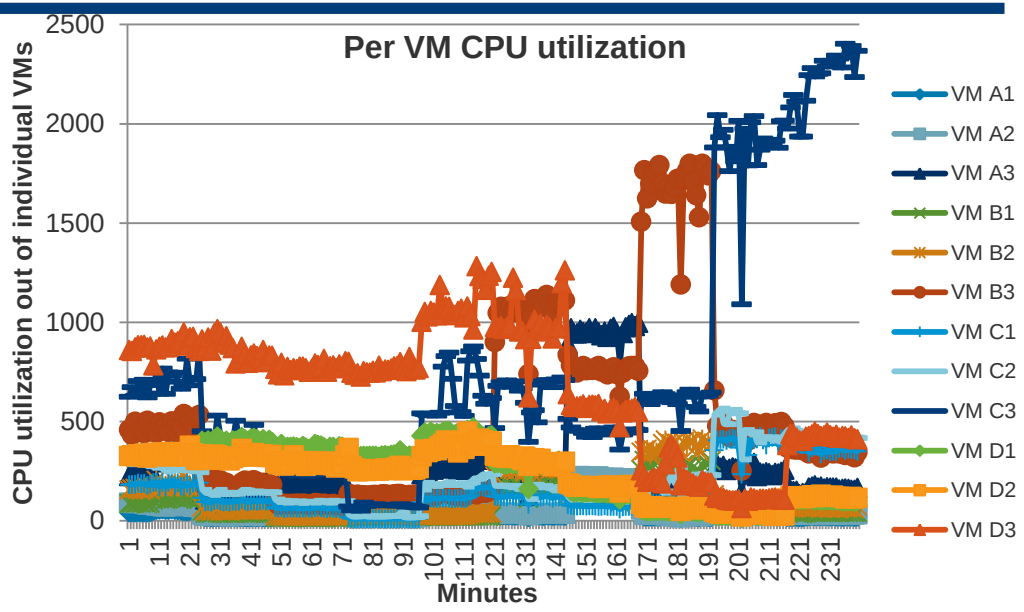
Throughput and CPU utilization of 12 VMs with static load

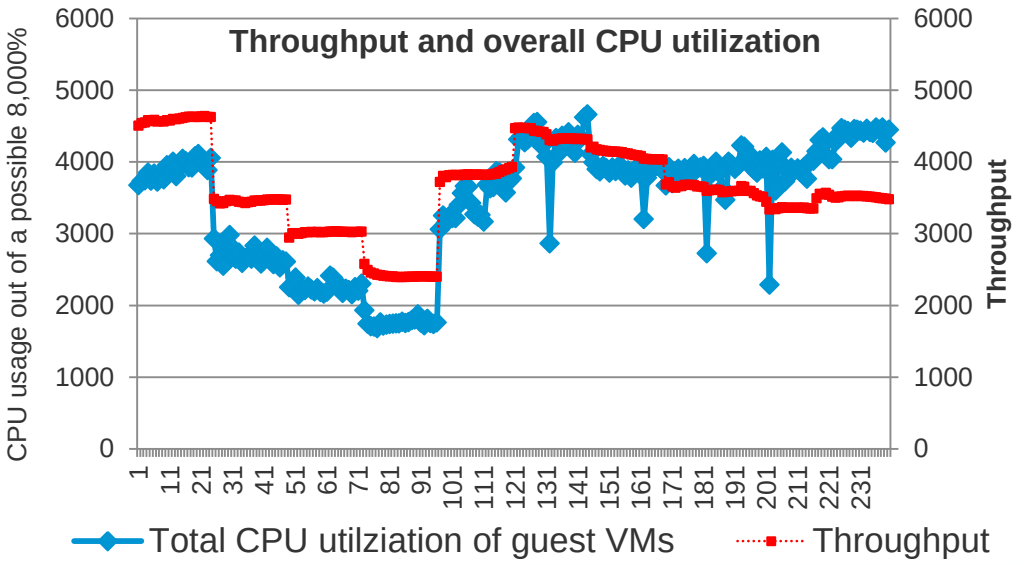
- **No elasticity**
  - Static load to all VMs over the 2 hours
- **Throughput droped with time as CPU util% rises**
  - Something wrong with VM C3. It utilization keeps rising, even above VM D3, despite static load
  - Early results. Our databases are better tuned now
- **Server 85-95% utilized**



Throughput and CPU utilization of 12 VMs with elasticity

- **Hypervisor has to react to 16X variation in load**
  - Change how much CPU is allocated to each VM
- **Overall CPU util% and throughput matches the static case in Phases 1 and 6 (no I/O bottleneck)**
- **The dips are the sign of the benchmark doing its job!!**
  - Caused by storage for a VM getting overwhelmed
  - When a VM's storage can't keep up, its throughput drops
  - Kit maintains the ratios between all Groups, so overall throughput drops





Status of the kit and of the benchmark

- **Functional specification in good shape, waiting for kit completion**
- **End-to-end kit running complete TPC-E workload**
- **Kit running all novel TPC-V functions, waiting for rewriting 2 transactions**
- **Would like to see more internal TPC prototyping before releasing to the wild**
- **Much PostgreSQL tuning remains**
  - Both CPU cycles and IOPS/tran are over 2X of commercial databases
  - Will engage the community
  - Is the community interested in matching commercial databases?
- **Will consider engaging groups outside TPC in the development process**
  - Not typical for the TPC
  - But we will make the case for it if we can demonstrate serious commitment

Future directions

- **Complete the kit!**
  - Make it available for prototyping
- **Complete the spec**
  - Kit will become publicly available when the benchmark is released
  - Need to release this benchmark in 2014
- **Single-system virtualization no longer exciting!**
- **Can we model a cloud-like, multi-server config?**
  - The benchmark and the kit already deal with elastic load to 1 server. Extend this to elasticity and migrations across multiple servers
  - Intuitively, a simple step from where we are
  - But need to come up with migration scenarios w/o too much *choreographing*
  - Avoid deep pocket escalation wars with bigger and bigger clusters
  - Add deployment and provisioning

Questions?