1.Processors

a) Today's commodity processors have 1 to 32 cores, with some more exotic processors boasting 72-cores, and specialized GPUs having 5000+ CUDA-cores. About how many cores/threads are expected to be in future commodity processors in the next five years?

It is difficult to predict that how many cores are going to be in a processor in next five years

preciously because adding more core does not necessary add extra value in terms of processing power after certain limit, but if we take 10% YOY improvement , the number should be around 120 cores. Given that Distributed computing has become ubiquitous , there may not be any necessity of these numbers of cores in a processor. For Data Centres it may still make some sense but for personal computing apart from gaming (which is again mostly driven by GPU power) user may not be able to take any advantage of this extra power.

b. How are these future processors going to look or be designed differently than today’s processors?

As moor’s law comes to an end , it will be difficult to put more and more transistors on a single silicon chip, if some other material such as graphene cannot be used to make newer

Chips, then some other architecture has to be followed. The problem with von Neumann architecture is that memory and processing is separate entity and memory speed and processing speed differs greatly, also data which is required for faster processing may be

Located at the next core or may be twenty meters away in the next rack which has to travel

through switches to reach the processor, so an architecture which consolidate memory and

processing at the same chip could be exploited. One such promising architecture could come from **Memristive devices for computing.** Memristive devices are electrical resistance switches that can retain a state of internal resistance based on the history of applied voltage and current.

Also **neuromorphic devices** or **quantum computers** can play a role in making HPC even faster or more parallel.

**neuromorphic chips** are designed like human brain and they do not follow von Neumann architecture , rather they follow neural network like pattern to do calculation , one such chip

announced by Intel is **LOIHI**. Neuromorphic processor features a total shakeup in terms of architecture. Intel have designed this processor to mimic a human brain, with an ability to learn and grow independently. This revolutionary approach features artificial neurons and pathways to mimic the circuits and function of our brain. Krzanich, Intel’s CEO, highlighted that their prototype chip, LOIHI, had learned basic object recognition, independently, within a matter of weeks.

c. What are the big challenges they need to overcome?

The problem with scaling up transistors is that as manufacturers put more and more transistors , the size of the transistors become very small, so small that eventually it will hit the size of a single silicon oxide atom beyond which it cannot be scaled down. So new material has to be found.

The conventional way of scaling cpu speed is also making the clock faster , but that will also incur tremendous hitting problem which cannot be solved with conventional cooling method. In face that is the sole reason why manufacturers preferred more cores than higher clock speed processors.

Cores also cannot be increased indefinitely because after certain point there would be no ROI as programmes would not be able to take advantage of them , also data locality for cores to do the processing is much more important after certain point , otherwise processors would simply seat idle as long as data is not available from memory.

d. Describe what a core and hardware thread is on a modern processor, and the difference between them?

Core is an independent processing unit inside a processor which has its own registers and ALU, control unit etc. Hardware thread is a set of instructions which is running inside that core. A core can switch between two threads while executing , by interleaving their instructions or by switching between threads when one is blocked waiting for memory or something. Cores that have this ability are called “SMT” or simultaneous multithreading, or hyperthreaded capable. They have enough registers to hold the active state of at least two threads.

e. What type of workloads are hardware threads trying to improve performance for?

There is no hard and fast answer, but most of the time we will not see any advantage for systems where the workflow/calculation is sequential. If however the problem can be broken down into tasks that can be run in parallel (or the problem itself is massively parallel [as some mathematics or analytical problems are]), we can see large improvements.

Some examples

* **Image processing** can often be done in parallel (e.g. split the image into 4 and do the work in 1/4 of the time) but it depends upon the algorithm being run to see if that makes sense.
* **Rendering of animation** (from 3DMax,etc.) is massively parallel as each frame can be rendered independently to others -- meaning that 10's or 100's of computers can be chained together to help out.
* **GUI** programming often helps to have at least two threads when doing something slow, e.g. processing large number of files - this allows the interface to remain responsive whilst the worker does the hard work (in C# the BackgroundWorker is an example of this)

f. Compare GPU and CPU chips in terms of their strength and weakness. In particular, discuss the trade-offs between power efficiency, programmability and performance.

**GPUs are generally more power efficient more than CPUs, but the comparison is not that straight forward.** GPUs optimized for power consumption have better power efficiency than CPUs for workloads suited to the GPU architecture. The super high-performance GPUs are still big power hogs, like Tesla K40+ which, while faster, consumes almost triple the power per GFlop.

**Let’s take an example.**

**Power** :CPU cores send data over a much longer path than GPU cores. If data travels 2 cm path between two CPU cores, then it travels only 0.5 cm between two cores in a GPU .

Not only travel length of data but also broadcasting of it helps too. When you need to make all cores have the same data at a given time, CPU does read from all cores. Waste of time and heat. GPU on the other hand, uses its broadcasting ability to clone same data to all cores in a single cycle.

GPU has addressable shared memory. This means you can control paths of data instead of relying on an implicit control by cache (L1,L2 etc) to shorten them even further. GPU supports complex data paths within each multiprocessor . since this is hardware-handled, you don’t lose as many cycles but a CPU would. GPU groups reads and writes before issuing them.

Now the rest of the energy (besides the control logic and cache) goes to computing on ALUs and FPUs.

this means more of the energy is spent on “computing” in a GPU while only a fraction of it goes to “computing” on a CPU.

When whole chip is used at the same time (by pipelining and asynchronous computing), GPU does more work per Joule of energy while CPU does the first job quicker. Making first job quicker is not as important as throughput in a GPU. Since GPU generally runs “1 instruction, all data” type of workloads, higher percentage of memory bandwidth is saved for data. But in CPU, half of bandwidth can go to instructions themselves.

Lastly, a new GPU can have 256 x 32bit private registers “per pipeline” that is much more than AVX gives (32) or SSE has (16) per pipeline. This reduces memory dependency even more, leaving more energy to computing part.

**Programmability: In terms of programmability any CPU and GPU can both be programmed, but the programming paradigm differs greatly**. In case of CPU normal c , java, python code

Would work for engineering application , such as writing code for an OS or a browser,

But for a GPU : CUDA, Open GL , DirectX has to be used.

Generally , we get programmable vertex and pixel shaders that allow execution of code directly on the GPU to manipulate the buffers that are to be drawn. These languages (i.e. OpenGL's GL Shader Lang and High Level Shader Lang and DirectX's equivalents ), has C style syntax. These shader languages give an immense amount of power to manipulate what gets drawn at a per-vertex or per-pixel level, directly on the graphics card, making things like shadows, lighting, and bloom really easy to implement.

These days GPUS are also used for programming BIG Data , neural network applications due to parallel nature of these technologies. The code can still be written in general purpose language such as python or R but can be executed in GPU. Before the advent of GPU these codes were generally executed in special purpose hardware called stream processor, such as IBM’s deep blue.

**Performance** : In terms of performance also it is not apple to apple comparison , as these two technologies operate in different spaces. These days CPUs are increasing number of cores , so that many parallel calculation can be done but still while rendering 3D graphics GPUs leave CPUs to dust.

As many mathematical calculations of parallel in nature can be done in GPUs much faster , GPUs are more and more used in scientific computing while CPUs are more suited for workloads where much of the work is engineering related such as more control flow , branch prediction etc, which simply cannot be done well in GPU.

g. Why do we not have processors running at 100GHz today (as might have been predicted in 2000)?

The reason is with more clock speed comes more heat, and sometime the heat is so high specially when the clock speed is 100GHz the cooling solution needed for these kinds of processor would not be trivial and would drive the cost of ownership for this kind of processors. Also with multicore processor same thing can be achieved with lower clock speed thus with less heat and more efficient power consumption.

2. Threading

a. Why is threading useful on a single-core processor?

Threading is useful in a single core processor because it may so happen that one thread is waiting for some IO operation to be finished , and this could be network IO or storage IO , in the meantime another thread could run and finish some of its task. In fact this is the same reason why python threading is useful , as we know that python interpreter is pinned to one core of any processor , so generally it may seem like threading may not be useful, but that is not the case precisely because of aforementioned reason.

b. Identify what a thread has of its own (not shared with other threads):

Thread ID is one thing which is not shared with other thread.

Generally we call a function which should execute under a thread , in that case functions local variable and code should not be shared with other thread, so a portion of the code area and data area is not shared.

c. What is the advantage of OpenMP over Pthreads?

OpenMP is convenient because it does not lock the software into a pre-set number of threads. This kind of lock-in poses a big problem for threaded applications that use lower-level APIs such as Pthreads or Win32. How can the software written with those APIs scale the number of threads when running on a platform where more processors are available? One approach has been to use threading pools, in which a bunch of threads are created at program start up and the work is distributed among them. However, this approach requires considerable thread-specific code and there is no guarantee that it will scale optimally with the number of available processors. With OpenMP, the number need not be specified.  
  
OpenMP’s pragmas have another key advantage: by disabling support for OpenMP, the code can be compiled as a single-threaded application. Compiling the code this way can be tremendously advantageous when debugging a program. Without this option, developers will frequently find it difficult to tell whether complex code is working incorrectly because of a threading problem or because of a design error unrelated to threading.  
  
Should developers need finer-grained control, they can avail themselves of OpenMP’s threading API. It includes a small set of functions that fall into three areas: querying the execution environment’s threading resources and setting the current number of threads; setting, managing, and releasing locks to resolve resource access between threads; and a small timing interface. Use of this API is discouraged because it takes away the benefits provided by the pragma-only approach. At this level, the OpenMP API is a small subset of the functionality offered by Pthreads. Both APIs are portable, but Pthreads offers a much greater range of primitive functions that provide finer-grained control over threading operations. So, in applications in which threads have to be individually managed, Pthreads or the native threading API (such as Win32 on Windows) would be the more natural choice.

d. Do more threads always mean better performance?

No.

1. Multiple threads might allow you to do things in parallel, if the CPU has more than one core available. So in an ideal world, e.g. calculating some primes, might be 4 times faster using 4 threads, if the CPU has 4 cores available and the algorithm work really parallel.
2. If you start more threads as cores are available, the thread management of your OS will spend more and more time in Thread-Switches and in such case your efficiency using your CPU(s) becomes worse.
3. If the compiler, CPU cache and/or runtime realized that you run more than one thread, accessing the same data-area in memory, it operates in a different optimization mode: As long as the compile/runtime is sure that only one thread access the data, it can avoid writing data out to external RAM too often and might efficiently use the L1 cache of your CPU. If not: It has to activate semaphores and also flush cached data more often from L1/L2 cache to RAM.

It is better to follow the below guidelines

* If possible use single threaded, shared-nothing processes to be more efficient
* If threads are required, decouple the shared data access as much as possible
* Don't try to allocate more loaded worker threads than available cores if possible

e. Is super-linear speedup possible? Explain why or why not.

Yes and No.

super linear speedup comes from exceeding naively calculated speedup even after taking into account the communication process (which is fading, but still this is the bottleneck).

For example you have serial algorithm that takes 1𝑡 to execute. You have 1024 cores, so naive speedup is 1024𝑥, or it takes 𝑡/1024, but it should be calculated , taking into account memory transfer, slight modifications to algorithm, parallelisation time.

So speedup should be lower than 1024x, but sometimes it happens that speedup is bigger, then we call it superlinear.

From vast amount of places: cache usage (what fit into registers, main memory or mass storage, where very often more processing units gives overall more registers per subtask), memory hit patterns, simply better (or a slight different) algorithm, flaws in the serial code.  
For example random process that searches space for result is now divided into 1024 searchers covering more space at once so finding solution faster is more probable.  
There are by-products (if you swap elements like in bubble sort and switch into Gpu it swaps all pairs at once, while serial only up to point).

On the distributed system communication is even more costly, so programs are changed to make memory usage local (which also changes memory access, divides problem differently than in sequential application).

And the most important, the sequential program is not ideally the same as parallel version - different technology, environment, algorithm etc. so it is hard to compare them.

Theoretically speedup can never exceed the number of processing elements 𝑝.

If the best sequential algorithm takes 𝑇𝑠 units of time to solve a given problem on a single processing element, then a speedup of 𝑝 can be obtained on 𝑝 processing elements if none of them spends more than time 𝑇𝑠/𝑝.

A speedup greater than 𝑝 is possible only if each processing element spends less than time 𝑇𝑠/𝑝 solving the problem.

In this case, a single processing element could emulate the 𝑝 processing elements and solve the problem in fewer than 𝑇𝑠 units of time.

This is a contradiction because speedup, by definition is computed with respect to the best sequential algorithm.

f. Why are locks needed in a multi-threaded program?

Locks are needed in a multithreaded program to make sure that shared data between threads are not manipulated out of order. Let’s say that there is a program which increment number *i*  1 to 10000. If we create two thread to do the same job meaning call on the same function which increment the value , then we would be able to see that *i* is incremented either out of order haphazardly. Instead of that if we create *i* as a critical section variable on which a lock has to be acquired by both the treads before doing some operation , then *i* would be incremented properly and not out of order.

Above is a very simple scenario which demonstrate that locks are needed in case of multithreaded programming so that critical section variable’s sanity is preserved across multiple threads.

g. Would it make sense to limit the number of threads in a server process?

Number of threads should not be limited in a server process . If the application needs to be ported on a different physical server which has more capacity in terms of processor hardware then limiting number of threads will actually stop the application from benefitting off the extra hardware capacity , but in that case OpenMP or ThreadPool can be used so that server program can take advantage of the extra hardware.

**3. Network**

a. A user is in front of a browser and types in www.google.com, and hits the enter key. Think of all the protocols that are used in retrieving and rendering the Google logo and the empty search box. Describe the entire sequence of operations, commands, and protocols that are utilized to enable the above operation.

1. You type **google.com** into address bar in your preferred browser.
2. The browser **parses the URL** to find the protocol, host, port, and path.
3. It **forms a HTTP request** (that was most likely the protocol or it could be HTTPS or more recently HTTP/2)
4. To reach the host, it first needs to **translate** the human readable host **into an IP number**, and it does this by doing a **DNS lookup on the host.**
5. Then a **socket needs to be opened** from the user’s computer to that IP number, on the port specified (most often port 80)
6. When a connection is open, the **HTTP request is sent** to the host
7. The host **forwards the request** to the server software (most often Apache) configured to listen on the specified port
8. The **server inspects the request** (most often only the path), and **launches the server plugin needed** to handle the request (corresponding to the server language you use, PHP, Java, .NET)
9. The plugin gets access to the full request, and starts to prepare a HTTP response.
10. To construct the response a **database** is (most likely) **accessed**. A database search is made, based on parameters in the path (or data) of the request
11. Data from the database, together with other information the plugin decides to add, is **combined into a long string** of text (probably HTML).
12. The plugin **combines** that data with some meta data (in the form of HTTP headers), and **sends the HTTP response** back to the browser.
13. The browser receives the response, and **parses the HTML** in the response.
14. A **DOM tree is built** out of the HTML
15. **New requests are made** to the server for each new resource that is found in the HTML source (typically images, style sheets, and JavaScript files). Go back to step 3 and repeat for each resource.
16. **Stylesheets are parsed**, and the rendering information in each gets attached to the matching node in the DOM tree
17. **Javascript is parsed and executed**, and DOM nodes are moved and style information is updated accordingly
18. The browser **renders the page** on the screen according to the DOM tree and the style information for each node
19. **You** **see** the page on the screen

**4. Power**

a. Why power consumption is critical to datacentre operations?

Power consumption represents a big portion of the Opex in data centre operations. More over if power consumption is high it can be an indicator of inefficient performance of some application or bad cooling solution which may result into higher MTBF of components of servers which in turn increases Capex of the organization. These days it is also important to keep the power consumption low and may be generate the power from complete renewable energy sources which is done in case of Facebook data centres.

b. What is dynamic voltage frequency scaling (DVFS) technique?

Dynamic voltage and frequency scaling (DVFS) is the adjustment of power and speed settings on a computing device’s various processors, controller chips and peripheral devices to optimize resource allotment for tasks and maximize power saving when those resources are not needed.

DVFS allows devices to perform needed tasks with the minimum amount of required power.  The technology is used in almost all modern computer hardware to maximize power savings, battery life and longevity of devices while still maintaining ready compute performance availability.

An unused smartphone, for example, should revert to a low-power mode, barring interference from applications and spyware. Multimedia requires more power, so the device reaches a higher power state and creates more heat during heavier processing such as video and gaming. Were it not for DVFS, many devices that are passively cooled would require active cooling. However, the noise, bulk and power consumption required by active cooling makes it impractical for smaller devices. DVFS helps maintain operable parameters with increased mobility.  
  
DVFS is not reserved for mobile technology alone. Desktops, servers and virtual environments also benefit from the power savings of DVFS. In VMware vSphere, DVFS allows host CPUs to dynamically change power states when resource demands are low to reduce a host's energy consumption. CPU frequency and voltages are lowered and raised based on demand from virtual machines (VMs).

While more important in environments with large numbers of computers, like offices, server farms and data centres, individual users are also paying more attention to power consumption. DVFS can also scale upwards to increase performance. Both AMD and Intel feature similar technology, often referred to as “turbo mode” for short runs of speeds beyond what the device could maintain indefinitely, at least with default cooling.

c. If you were to build a large $1B data center, which would require $50M/year in power costs to run the data center and $50M/year in power costs to cool the data center with traditional A/C and fans. Name 2 things that the data center designer could do to significantly reduce the cost of cooling the data center?

They could make the server farm near some ocean or large waterbody , so that it is earlier to dissipate the heat into the ocean via viaduct or large tubes and fans. After sometime this arrangement would offer free cooling.

They could also use the heat generated by the servers to warm up office spaces which in turn would save a lot of power bill.

d. Is there any way to reduce the cost of cooling in (C)? If yes, how low could the costs go? Explain why or why not?

The ways to reduce the cost have been explained earlier . In the first scenario the cooling become free after certain time period. There are instances when companies have shifted their data center into places like Minneapolis, so that natural low temperature could actually help them to reduce the cost of cooling to zero. It may be expensive to build such a facility and that would incur a lot of Capex but over a period of time as the Cooling becomes free that cost would be recovered.

Microsoft is also experimenting with under water data canters to reduce the cost of cooling and also operate their data centres with maximum efficiency.

**5. Storage**

a. If a manufacturer claims that their HDD can deliver sub-millisecond latency on average, can this be true? Justify your answer?

No this would not be true. For this to be true the rotation speed has to be extremely high in the range of 90000 to 100000 rpm. In a high end HDD spinning at the speed of 15000 rpm gives 2ms rotational latency apart from that to read the data there is seek latency + some overhead (which is always there), so to provide sub milliseconds latency all of these measurement has to go down significantly which is very difficult. For a HDD to spin more than 15000 rpm it would consume too much power for it to be reasonable and everything follows from there.

b. Explain why flash memory SSD can deliver better performance for some applications than HDD.

Applications which mostly do random read write is better suited for flash SSD then HDD. Due to the absence of moving part wear and tear in case of random read write is also much less in flash SSD than HDD.

c. What types of workloads benefit the most from SSD storage?

Random read write benefits most from SSD.

d. If a manufacturer claims they have built a storage system that can deliver 1 Terabit/second of persistent storage per node, would you believe them? Justify your answer to why this is possible, or not. Make sure to use specific examples of types of hardware and expected performance.

Seagate Technology and storage system company, Newisys (a division of Sanmina Corp.) announced that 21 Newisys NSS\_2601 with dual NSS-HWxEA Storage Server Modules deployed with Seagate’s SAS 100.2 SSD drives can be combined in a single 42U rack to provide block I/O performance of 1 TB/s with 5 PB of storage.

This can be achieved by distributing the write into multiple very small blocks and writing the data into multiple SSDs parallelly .

e. In this problem you are to compare reading a file using a single-threaded file server with a multithreaded file server. It takes 16 msec to get a request for work, dispatch it, and do the rest of the necessary processing, assuming the data are in the block cache. If a disk operation is needed (assume a spinning disk drive with 1 head), as is the case one-fourth of the time, an additional 32 msec is required. What is the throughput (requests/sec) if a multi-threaded server is required with 4-cores and 4-threads, rounded to the nearest whole number?

Each core executs each thread parallelly. 3 request served from block cache by 3 thread within 16 msec. The thread in the last core blocks for IO for 32 msec, in that time 2 more request is served .

So in a block of 16 msec time 3 request is servef from 3 core. So , 1 second = 1000 msec,

So, 1000/16 \*3= 62.5\*3 = 187.5 + (48(16+32) msec to serve one delayed request)

So, 187.5 + 1000/48 = 187.5 + 20.83 = 208.33 or ~ 208 request/sec.

6a)

100TB of data.

random access query which touches 10% of data that is 10TB.

200 cores with 2.0GHz

processor x86

10% in main memory. so RAM should be 10TB check again?

storage 100TB

100GbE switch.

power : $0.01KWH cooling : 0.01KHW

administration : 10000$ /Year

solution for 5 years including hardware , power, cooling , administration.

Spark:

Model :

Intel® C612 Chipset - 10x SATA and 2x NVMe - Dual Intel® 10-Gigabit Ethernet (RJ45) - IPMI 2.0 with Dedicated LAN Port

2\*Ten-Core Intel® Xeon® Processor E5-2630L v4 1.80GHz 25MB Cache (55W)

16\* 64GB PC4-19200 2400MHz DDR4 ECC Registered Load-Reduced DIMM = 1TB

5\* 2.0TB SATA 6.0Gb/s 7200RPM - 2.5" - Seagate Exos 7E2000 Series (512e)

1 Intel® 10-Gigabit Ethernet Converged Network Adapter X520-DA2 (2x SFP+)

5 Year Advanced Parts Replacement Warranty and NBD Onsite Service

<https://www.thinkmate.com/system/rax-xt10-2160v4-nvme>

Total cost per server = $ 21,300

TCO = 21,300 \* 10 = $213000

estimated power is 351.9 watts /1000 = 0.3591 KWH

BTU/h = 1201

$0.10KHW \* 0.3591 or 10 cents \* 0.35 = 3.5 cents per hour \* 10 servers = 35 cents per hour for all the servers \* 5 years

or , 35 \* (5\*365\*24) = 35 \* 43800 = 1533000 cents = $15330 power bill + $15330 cooling bill

or , 30,660 total

**Total cost = $213000 + $30,660 = $243660 (Total cost in 5 years) + cost of switch (which is not available in ThinkMate)**

My SQL

———

2 sql node, 2 cluster management node, 6 data node.

6 \* 1.5 TB = 9TB

Rest 1 TB = 450 GB each on 2 sql node which is 900 GB Rest 100 GB divided between 2 management node.

Model

———

Intel® Server System R2308WTTYSR - 2U - 8x 3.5” SATA/SAS - Dual 10-Gigabit Ethernet - 24x DDR4 - 1100W

2\* Fourteen-Core Intel® Xeon® Processor E5-2660 v4 2.00GHz 35MB Cache (105W) = 28 cores \* 6 = 168 cores

24 \* 64GB PC4-19200 2400MHz DDR4 ECC Registered Load-Reduced DIMM =1.5TB of primary memory = 9TB memory

2\*6.0TB SATA 6.0Gb/s 7200RPM - 3.5" - Ultrastar™ DC HC310 (512e)[ -418.00 ] = 12 TB \* 6 = 72 TB HDD

1\*Intel® 10-Gigabit Ethernet Converged Network Adapter X520-DA2 (2x SFP+) \* 6 = 60 GbE speed

5 Year Advanced Parts Replacement Warranty and NBD Onsite Service

TCO of one machine : $28013 , we need 6 such machines = so 6 \* 28013 = $168078

Total wattage : 455.5 watt / 1000 = 0.455 KHW \* 10 cents = 4.55 cents per hour \* (5\*365\*24) = 4.55 \* 43800 = 199290 cents = $1992 \* 6 servers = $12000

BTU/h = 1554

+ 2 sql node

Model: AMD® SR5670+SP5100 Chipset - Dual Intel® Gigabit Ethernet - 6x SATA - IPMI 2.0 with LAN

2\*Twelve-Core AMD Opteron™ Model 6338P - 2.3GHz 16MB Cache (99W TDP) = 24 cores

16\*32GB PC3-12800 1600MHz DDR3 ECC Registered Load-Reduced DIMM = 512 GB \* 2 = 1TB

2\*6.0TB SATA 6.0Gb/s 7200RPM - 3.5" - Ultrastar™ DC HC310 (512e) = 12 TB \* 2 = 24TB

1\*Intel® 10-Gigabit Ethernet Converged Network Adapter X520-DA2 (2x SFP+) = 10GbE \*2 = 20GbE

5 Year Advanced Parts Replacement Warranty and NBD Onsite Service

TCO of one machine $11915 , we need 2 such machine = $23830

Total Watt : 377.8 /1000 = 0.377 \* 10 cents = 3.77 \* (43800) = 165126 cents = $1651 \* 2 = $3302

+ 2 management server

Model :

AMD® SR5670+SP5100 Chipset - Dual Intel® Gigabit Ethernet - 6x SATA - IPMI 2.0 with LAN

2\*Quad-Core AMD Opteron™ Model 6308 - 3.5GHz 16MB Cache (115W TDP) = 8 cores \* 2 = 16 cores

8\*4GB PC3-14900 1866MHz DDR3 ECC Registered DIMM = 32 GB RAM \* 2 = 64 GB of RAM

2\*1.0TB SATA 6.0Gb/s 7200RPM - 3.5" - Ultrastar™ DC HA210 (512n) = 2TB \*2 = 4TB HDD

1\*Intel® 10-Gigabit Ethernet Converged Network Adapter X520-DA2 (2x SFP+) = 20GbE

5 Year Advanced Parts Replacement Warranty and NBD Onsite Service

TCO one machine : $4198 \* 2= $8396

Tota watt= 354.4 watt / 1000 = 0.354 \* 10 cents = 3.54 \* (43800) = 155052 cents = $1550 \* 2 = $3100

Total Cost = $168078 + $23830 + $8396 = $200304

$12000 + $3302 + $3100 = 18402 \* 2 (cooling cost) = $36804

**Total cost in 5 years = $200304 + $36804 = $237108 + cost of switch (which is not available in ThinkMate)**