Cloud Algorithms and Mathematics

Barry Denby

Griffith College Dublin

April 30, 2020

Cloud Algorithms and Mathematics

- In this course thus far we have looked at the general theory of cloud computing systems
- We have covered areas as diverse as virtualisation, storage, security, and code optimisation
- In this lecture and the following lectures we will go deeper into the theory and explore some definitions and mathematics of cloud computing
- Implemented in most cloud systems under the hood
- Most of this material will be heavily based off Cloud Computing: Theory and Practice

Cloud Algorithms and Mathematics

- We will cover material related to
  - Parallelism in the cloud
  - Scheduling
  - Resource Management
- While this looks scattered there is a lot of material to choose from

Speedup of parallel computations

- Cloud computing environments are inherently parallel in nature
- Large problem sets can be run on the cloud with multiple virtual machines each processing a subset of the data
- Allows you to solve problems that demand more resources than those provided by a single system
- In the general case the speedup $S(N)$ of a parallel solution is $S(N) = \frac{T(1)}{T(N)}$

Speedup of parallel computations

- $T(1)$ is the execution time of the single threaded solution
- $T(N)$ is the execution time of the parallel solution
- Where N is the number of participating nodes in the parallel computations
- This is an important value that tells you both how scalable your algorithm is and how much performance it is generating

Speedup of parallel computations

- If $S(N) = 1$ then it means the parallel solution is no faster than the single threaded solution
- If $S(N) < 1$ then it means the parallel solution performs worse than the single threaded solution
- if $S(N) > 1$ then it means the parallel solution performs quicker than the single threaded solution

Speedup of parallel computations

- If $S(N) = N$ then we have linear speed up
  - There is no performance loss in the parallel solution for each node added
- If $S(N) < N$ then we have sub-linear speed up
  - There is some performance loss when a new node is added
  - The most common case as we expect some performance loss with additional synchronisation and communication
- If $S(N) > N$ then we have super-linear speed up
  - There is extra performance gained by running in parallel
  - Ideal but this is a very rare occurrence

Amhdal's law

- Gives the potential speed up of a parallel computation
- Defines a percentage of the computation that cannot be parallelised α
- Law is expressed as $S = \frac{1}{\alpha}$
- Where S is the speedup and α is the fraction of code that cannot be run in parallel
- As alpha increases then potential speedup will decrease and vice versa

Amdhal's law

- To prove this we define (σ) to be the sequential segment and (π) to be teh parallel segment. As a result we can rewrite portions of our formulas as follows
- $T(1) = \sigma + \pi$, $T(N) = \sigma + \frac{\pi}{N}$, and $\alpha = \frac{\sigma}{\sigma+\pi}$
- By pluggins these into Amdhal's law and working with the equations we get (for large values of N)
- $S = \frac{1+\frac{(1-\alpha)}{\alpha}}{1+\frac{(1-\alpha)}{N\alpha}} == \frac{1}{\alpha+\frac{(1-\alpha)}{N}} \approx \frac{1}{\alpha}$

Gustafson's Law

- The problem with Amdahl's law is that it deals with a fixed problem size
- Realistically this will never be the case because, as more nodes are added to a parallel system there will be bigger datasets added to ensure speedup.
- Thus Gustafson modified Amdahl's law to account for a variable problem size.
- Gustafson's law is written as follows
- $S(N) = N - \alpha(N - 1)$

Gustafson's Law

- To prove this law we set the following
- $T(1) = \sigma + N\pi$, and $T(N) = \sigma + \pi$
- and plugging into the formula we get
- $S(N) = \frac{T(1)}{T(N)} = \frac{\sigma+N\pi}{\sigma+\pi} = \frac{\sigma}{\sigma+\pi} + \frac{\pi}{\sigma+\pi}$
- $= \alpha + N(1-\alpha) = N - \alpha(N-1)$

Coffman Conditions for concurrent deadlock

- In a system like the cloud there is the possibility with shared resources that deadlock can occur in the system
- Particularly if those shared resources are protected by locks to ensure serial access
- For deadlock to occur in a concurrent system there are four conditions that must be satisfied simultaneously
- These are known as the Coffman conditions

Coffman Conditions for concurrent deadlock

- Mutual Exclusion: At least one resource cannot be shared, only one process can access it at a time
- Hold and Wait: At least one process holds a resource and is waiting for other resources
- No preemption: The monitor process cannot force a process to release a resource
- Circular wait: P1 waits for P2 which waits for P3 waits for P4 ... PN waits for P1

Resource Management

- Resource management is core to any system as it affects the three basic criteria for evaluation of a system
  - Performance
  - Functionality
  - Cost
- Some functions may be too expensive or exhibit poor performance
- The strategies for resource management differ for all three cloud models

Resource Management

- Some problems like fluctuating workloads are shared by all three models
- Sometimes this can be predicted in advance e.g. seasonal loads
- Other times it will be unpredictable
- Requires spare available capacity
- And a monitoring system that can allocate resources to applications with little to no delay

Resource Management

- Unfortunately due to the size of the cloud, centralised control will not provide continuous service or performance guarantees
  - Great interest in autonomic policies
  - The idea being to get the system to manage itself
- There are many different policy areas that must be enforced and policed in a cloud system
- We will only scratch the surface and introduce some basic ways of determining who should get access to resources

Policies for Resource Management

- Cloud resource management policies can be loosely grouped into five classes
  - Admission Control
  - Capacity Allocation
  - Load Balancing
  - Energy Optimisation
  - Quality of Service
- Load Balancing and Energy Optimisation were talked about in much detail in an earlier lecture.

Admission Control

- The explicit goal of an admission control policy is to prevent the system from accepting workloads in violation of high-level system policies
- Not accepting work that would prevent current work from being completed
- Requires knowledge of the system global state
- Which in a system of cloud scale is obsolete at best

Capacity Allocation

- Allocation of resources to individual instances
- Locating resources is subject to multiple global optimisation constraints
- Requires a search of a very large search space
- However, individual system state can change very rapidly

Quality of Service

- Gives guarantees about the performance of all components in the cloud
- These guarantees should never be broken
- However, this is the most difficult to address (as we will see with scheduling later)
- But this is the most critical to cloud computing

Policies for Resource Management

- The problem here is that the already existing optimal solutions for these problems do no scale up to large systems
- They also only target single parts of resource management
- Also many have complex calculations that are too slow for a cloud to compute fully and in time
- Cloud needs algorithms that will address multiple areas at once

Mechanisms for Resource Control

- There are four basic implementations of resource management
- Control Theory: uses sensors and feedback to guarantee system stability and predict transient behaviour
  - However, this can only predict local not global behaviour
- Machine learning: a technique that could be applied to coordinate several autonomic system managers
  - Does not require a performance model of the system

Mechanisms for Resource Control

- Utility Based: requires a performance model and a mechanism to correlate user-level performance with cost
- Market-oriented/economic mechanisms:
  - Do not require a model of the system
  - Have auctions for bundles of resources
  - Simulates an economy for resources

Mechanisms for Resource Control

- Distinctions should be made between interactive and non interactive workloads
- The management techniques differ for both
- Interactive workloads need flow control and dynamic application placement
- Non-interactive workloads are focused on scheduling
- Trying to account for both types is a very difficult challenge

Scheduling Algorithms for Computing Clouds

- Scheduling is a critical component of cloud resource management
  - responsible for resource sharing and multiplexing on many levels
  - server $\rightarrow$ VMs $\rightarrow$ applications $\rightarrow$ threads etc
- Good scheduling algorithms must have the following properties
  - efficient, fair, and starvation-free
  - The objectives for the scheduler depend on the type of workloads submitted to the system the scheduler will manage

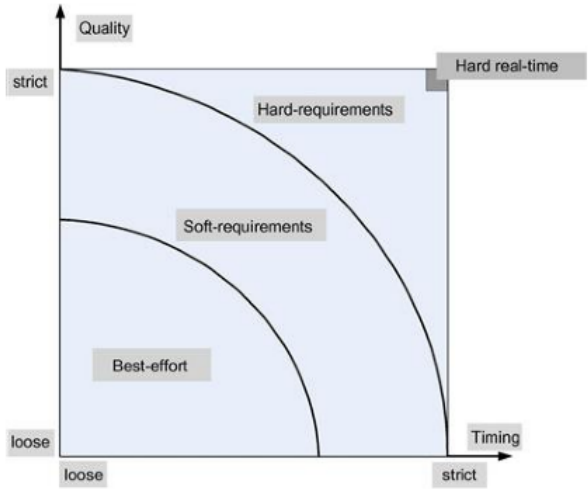
Scheduling Algorithms for Computing Clouds

- E.g. batch scheduler tries to maximise throughput and minimise turnaround time
  - Time between job submission and completion
- E.g. real-time system must meet deadlines and be predictable
  - There is a hard defined time limit for each task
- Schedules can either be preemptive or non-preemptive
  - Preemptive: may interrupt a low priority task to complete a high priority task
  - Non-preemptive: all tasks cannot be interrupted by the scheduler regardless of priority

Scheduling Algorithms for Computing Clouds

- Schedules must address two aspects of resource management
  - The quantity of resources that are allocated
  - When the resources are allocated
- Plotting these parameters on a graph leads to the figure on the next slide
  - Determines the different classes of schedules and their respective requirements
  - Best-effort, Soft-requirements, and Hard-requirements are the different classes

Scheduling Algorithms for Computing Clouds



Scheduling Algorithms for Computing Clouds

- Best-effort policies don't impose requirements on resource quantity or timing with respect to allocation of resources
- Soft-requirements require statistically guaranteed amounts of resources and timing constraints
  - They will be observed the majority of time but allows for cases where these guarantees can be broken
- Hard-requirements dictate that all limits on resource amounts and timing are respected
  - Nothing is allowed to break these limits

Fair Queuing

- Computing and communication are common operations on the cloud
  - Thus scheduling algorithms used in the cloud will try and link the two together
- Such scheduling algorithms must manage many quantities at the same time
  - Bandwidth: The amount of data each flow is allowed to transport
  - Timing: When the packets of individual flows are transmitted
  - Buffer space: Storage for packets allocated for each flow

Fair Queuing: FCFS

- The easiest solution to this is to use an FCFS scheduler
- However, it does not guarantee fairness as greedy flows can get more bandwidth
- Another option is to use a queue for each flow and service each flow in a round-robin manner
- However, this does not guarantee fairness of bandwidth allocation

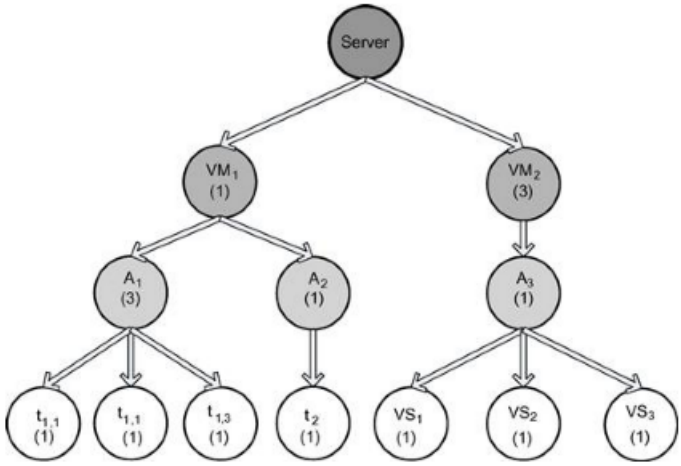
Fair Queuing: Start-time Fair Queuing

- An algorithm for allocating CPU bandwidth originally designed for multimedia applications
- The idea is to organise CPU bandwidth consumers into a tree like structure
- Root node is the processor, leaves are individual threads
- A scheduler acts at each level of the hierarchy

Fair Queuing: Start-time Fair Queuing

- The fraction of processor bandwidth B_i allocated to the intermediate node i is the following
- $\frac{B_i}{B} = \frac{w_i}{\sum_{j=1}^{n} w_j}$
- Where w_i is the weight of tree node i and the summation of w_j is the combined weight of all nodes in the tree at the same level of that node
- An example of such a tree is shown on the next slide

Fair Queuing: Start-time Fair Queuing



Fair Queuing: Start-time Fair Queuing

- The Tree that you see here has two virtual machines with three applications
- Each VM and application weighted and scheduled by the node above
  - E.g. Server decides VM's weighting and VM's decide the weighting of individual applications
- The tree is used to calculate the proportion of CPU time that is allocated to each thread
  - Gives a cascading calculation for example t(1,3) will get 1/3 of A(1)'s processing time which in turn gets 3/4 of VM(1)'s processor time
  - Roughly 5 to 6% of processor time

Fair Queuing: Start-time Fair Queuing

- When a VM is not active its bandwidth will be reallocated to other VMs that are active
- Similar for applications and threads
- If a thread is not runnable its allocation will be transferred to other threads