MapReduce in the Clouds

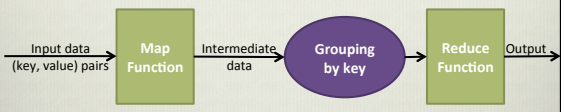
Hessah Alsaaran

Presentation #1
CS 655 – Advanced Topics in Distributed Systems
Computer Science Department
Colorado State University

MapReduce (MR)

- A programming model for parallel processing of big datasets, first introduced by Dean and Ghemawat [1].
- It provides a simple interface to the user that allows the user to process the data in parallel in a distributed system, hiding the details from the user.
- Its objective is to reduce the response time for processing large datasets.

MR in General



- There is a single master and multiple slaves for each MR job.
- The master distributes/schedules the map and reduce tasks to the slaves, and handles their failure.

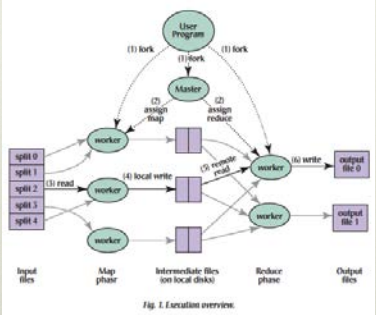


Fig. 1. Execution overview.

Figure 1 from [1]

Handling Slave Failures

(1) Non-Responding Slave

- Reinitialize its tasks and reschedule them on other available slaves.

(2) Straggler Slave

- Schedule backups of its tasks on other slaves.
- A task can be completed by either the primary or the backup slave, whoever finishes first.

- What about master failure?

Pulls and Pushes

Pulls

- Huge data volumes are generated continuously [1, 6]
  - They need to be processed/analyzed.
  - Increase in data size does not match increase in processing speed.
- →The need for large data parallel processing that decreases response time.

Pushes

- Open-source MR implementations. [5]
- Publically available cloud services. [2, 5, 6]
- The drop in computer prices.

When to use MR?

- Compared to parallel DBMSs: (Stonebraker et al. [5])
  - For datasets that are read only once.
  - When computations are complex with multiple data passes.
  - When the data is semi-structured, for example (key, value) pairs.
  - For quick jobs: quick setup and start-up time.
  - When your budget is low: free open-source implementations.

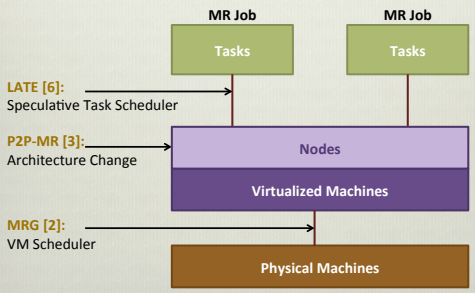
7

What is the problem?

- The dynamic and heterogeneous environment of the clouds:
  - All nodes are not equal. (different capabilities and generations) [6]
  - Even if equal, VMs compete on them. It can cause I/O contention. [4, 6]
  - Dynamic nodes (node churn and failures). [3]
- These problems may degrade the performance of MR in the clouds. [3, 4, 5, 6]

8

Three Approaches to Improve the Performance



9

First Approach

10

First Approach

- **Longest Approximate Time to End (LATE) Scheduler, by Zaharia et al. [6].**
  - A task scheduler for speculative execution.
  - The authors are trying to solve an inefficiency of Hadoop (an MR implementation): when scheduling a speculative task, it assumes that all stragglers are equal.
  - Their solution is:
    - Rank the tasks based on their estimated finish time.
    - Schedule the tasks on fast nodes.

11

Hadoop

- A single master and multiple slaves.
- The MR job is divided into a set of map and reduce tasks.
- Each slave has a set of task slots (default is 4 slots, 2 for each type) to concurrently run its tasks.
- When a slave has an empty task slot, it asks the master for a task. A task is chosen by the scheduler and assigned to this slave.

12

Hadoop's Scheduler

- The tasks considered for assignment are ordered as follows:
  1) Failed tasks (highest priority)
  2) Non-running tasks
     - For map tasks -> Choose first the tasks for data that is local at the requesting slave.
  3) Speculative tasks
     - They are all equal, but data locality is also considered here for assignment.

13

Speculative Tasks Identification

- Every task has a progress score in the range [0,1].
- **Map task:**
  - Progress score = the fraction of input data read
- **Reduce task:**
  - The task is divided in 3 phases:
    - Copy phase: map outputs are copied to local storage.
    - Sort phase: grouping by key.
    - Reduce phase: executing the reduce function.
  - Each phase accounts for 1/3 of the progress score, and each score is the fraction of data processed.

14

Speculative Tasks Identification

- **Example:**
  - If the reduce task has performed half of the **copy** phase:
    - Progress score = (1/2).(1/3) = (1/6)
  - If the reduce task is in the middle of the **reduce** phase:
    - Progress score = (1/3) + (1/3) + (1/2).(1/3) = (5/6)

15

Speculative Tasks Identification

- The average progress score for each task type is computed.
- Threshold = average progress score – 0.2
- A task is marked as a straggler and considered for speculation execution, if:
  - It has been running for more than 1 minute.
  - Its progress score is less than the threshold.

16

Hadoop's Scheduler Assumptions

1. The tasks progress rate at each slave are roughly the same.
2. A task's progress rate is constant.
3. Performing speculative tasks in idle slots of slaves does not incur additional cost.
4. Each phase in the reduce task represents 1/3 of its total time.
5. A straggler likelihood depend on its progress score.

17

Assumptions Invalidated

- These assumptions don't hold in a virtualized data center → heterogeneous environment.
- Not all slaves are equal
  - difference could be in the physical machines or the competing co-allocated VMs.
  - This invalidates the first two assumptions: progress rates are not equal.
  - Fixed threshold → too many speculative tasks.
  - All speculative tasks are equal → incorrect, they vary in progress achieved and progress rate.

18

Assumptions Invalidated

- The other 3 assumptions are invalid in both homogeneous and heterogeneous clusters.
- The assumption of no additional cost for performing speculative tasks on idle slots of slaves does not hold when resources are shared:
  - Ex: network, and disk I/O.
  - Needless speculation can reduce throughput.
  - Also not true in "pay-as-you-go" settings.

19

Assumptions Invalidated

- The assumption that each phase in the reduce task represents 1/3 of its total time.
  - The copy phase is the slowest.
  - After the copy phase, the task's progress score changes from 1/3 to 1 quickly → increases the average progress score.

20

Assumptions Invalidated

- **Example:** 30% of reducers finish, while the others are still in the copy phase.
  - Average progress score = (0.3).(1) + (0.7).(1/3) ~= 53%
  - The threshold jumped from (33 – 20 = 13%) to (53 – 20 = 33%)
    → all the remaining 70% of the reducers are considered for speculation.
    → task slots will be filled from an arbitrary selection.
    - True stragglers may have not been chosen.
    - Network overloaded with unnecessary copying.

21

Assumptions Invalidated

- The assumption that a straggler's likelihood depend on its progress score.
  - New and fast tasks can be chosen over old and slow tasks.
  - It should also consider progress rate.
  - Average progress score limit is 100%
    → Threshold = 80%.
    → Tasks with progress score > 80% can never be considered for speculation.

22

LATE Scheduler

- **Goal:** reduce the MR job response time.
- **Approach:**
  1. Prioritize the speculative tasks based on their estimated finish time.
  2. Schedule the tasks on fast nodes (avoid stragglers).
- **Estimating a task's finish time:**
  - Progress rate = progressScore / T
  - Remaining time = (1 – progressScore) / progressRate
  - Assumption: the task's progress rate is constant (may not hold).

23

LATE Scheduler

- **Determining which nodes are slow:**
  - A node's total performed work = sum of progress scores for both completed and in-progress tasks.
  - If total < SlowNodeThreshold → it is a straggler.
- **To reduce resources cost:**
  - speculativeCap: a cap on the number of running speculative tasks at a time.
  - Use the SlowTaskThreshold as an additional condition for running speculative tasks.
    - If there are only fast tasks that are running → no need to speculate them.

24

LATE Scheduler

- **Algorithm:**
  - When a node asks for a task
    - If the number of running speculative tasks < speculativeCap
      - If the node's total progress > slowNodeThreshold
        - Rank running tasks (that are not currently speculated, and have run for more than 1 minute) by estimated remaining time
        - Choose the highest-ranked task with progress rate < slowTaskThreshold, then send it the asking node.
- **Note:** it does not consider data locality such as in Hadoop.

25

LATE Scheduler

- **Setting parameters:** (by practice)
  - SpeculativeCap = 10% of available task slots.
  - slowNodeThreshold = 25th percentile of node progress.
  - slowTaskThreshold = 25th percentile of task progress rates.

26

LATE Scheduler

- **Some incorrect estimated times:**
  - The estimated remaining time for a task was based on the progress rate: (1 – ProgressScore) / ProgressRate.
  - This assumed constant progress rate per task, which may not hold.

27

Incorrect Estimated Times - Example

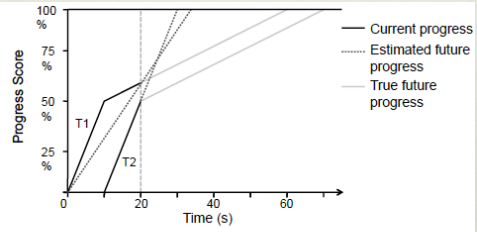


Figure 2: A scenario where LATE estimates task finish orders incorrectly.

Figure 2 from [6]

28

Incorrect Estimated Times

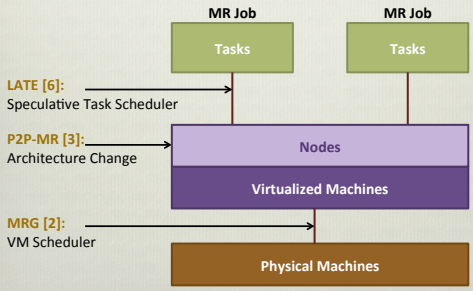
- **2 Situations:**
  - A task's progress rate slows down: caused the problem.
  - A task's progress rate speeds up: no problem here.
- This problem is not frequent in typical MR jobs:
  - Reduce tasks are slow in their first phase then speed up.
- A possible solution for the less typical MR jobs:
  - Account for each phase independently (per-phase progress rate)

29

Second Approach

30

Three Approaches to Improve the Performance



31

Second Approach

- **MapReduce Group Scheduler (MRG), by Kang et al. [2].**
  - A VM scheduler for scheduling the VMs on the physical machines (proposed as an extension to Xen).
  - **Inefficiencies** of Xen:
    - MR tasks are I/O-bound → increase context switch overhead
    - Boosting a VM frequently can cause scheduling delay of other VMs on the same machine.
  - **Observation:** For a MR job, its VMs have similar behavior.

32

MR Job Characteristics

- Most MR jobs are data-intensive → I/O-bound tasks.
- Frequent I/O requests from each VM
  - → Additional overhead on the VMM.
  - The VM will be put in the wait queue after an I/O request, and then back to the top of the run queue when the event is delivered by the driver domain.
    - → limits responsiveness to the other VMs.
- For a MR job, its VMs have similar behavior.

33

Xen's Credit Scheduler

- VMs are assigned credits used to schedule them on the CPU.
- Priority order: boost > under > over > idle.
- The run queue is sorted by the priorities.
- VMs loose some credit when de-scheduled after allocated time slice.
- dom0 (special driver domain - the layer between the VMs and the PM) handles all I/O processing.

34

Xen's Credit Scheduler

- If a VM request I/O, the request is passed to dom0 and the VM goes to the wait queue.
- When dom0 is scheduled on CPU, it handles any pending I/O requests.
- Upon completion of an I/O request, the VM is boosted in the run queue to handle the I/O.
- Scheduling dom0 too often incurs higher context switch overhead with fewer I/O handling per scheduling.

35

Example

- 4 domains are running on the same PM:
  - dom0: driver domain
  - dom1 and dom2 belong to a MR job, while dom3 belongs to another.

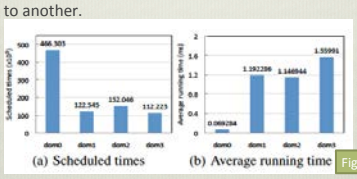


Figure 3 from [2]

  - The driver domain dom0 is scheduled after every de-scheduling of the VMs.

36

Example

In addition to unfairness, they can be considered as stragglers.

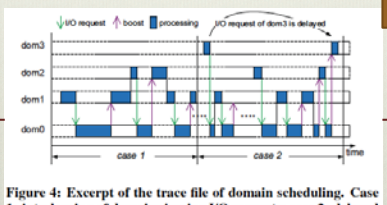


Figure 4: Excerpt of the trace file of domain scheduling. Case 1: interleaving of domains issuing I/O requests; case 2: delayed execution – the I/O request of dom3 is delayed

Figure 4 from [2]

If dom0 scheduling is deferred until after dom2, the I/O requests can be processed by a single scheduling. → Reduce context switch overhead.

37

MRG Scheduler

- **Idea:** use the knowledge about the groups for MR jobs.
- **To reduce context switch overhead:**
  - Instead of sorting the CPU run-queue only using FIFO within each priority, it groups by MR job too.
  - Defers dom0 to after a MR job group.

38

Sorting Example

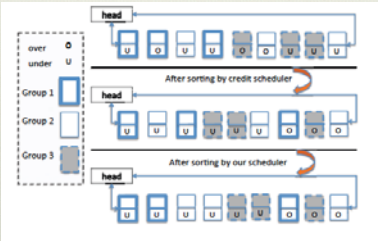


Figure 5: Sorting VCPUs in the run queue (We do not show dom0, boost, and idle state for simplicity.). The run queue is a doubly linked list. The scheduler picks up the VCPU to run on the PCPU from head of the run queue. Each VCPU has the priority of *over* or *under*.

Figure 5 from [2]

39

MRG Scheduler

- **Possible problem of deferring dom0:**
  - A group of VMs are not requesting any I/O.
  - Causes delay to the other I/O pending VMs.
- **Solution:**
  - Predict the I/O request likelihood of a VM.
  - It is based on running time of its last scheduling.
  - If it is less than a threshold -> more likely to request I/O next scheduling.

40

MRG Scheduler

- **To improve fairness:**
  - The CPU time allocated for a VM depends on its credit and its MR group credit.
  - It provides fair CPU time to each group, and then to each VM belonging to a group → hierarchical scheduling.
    - The scheduler keeps track of the running time of each group
    - Within each group's allotted time, only VMs of this group can run.
    - → tasks in the same group will progress around the same rate.

41

Hierarchical Scheduling

- **First level:**
  - Allocate credits for groups:
    - The weight of a group is the sum of each of its VMs credits.
    - User allocated credits.
      - VM credits are normalized as $C_g \times (C_i / \sum C_j)$
- **Second Level:**
  - Allocate CPU to to VMs in a group using their credits.

42

Example

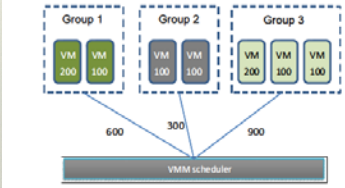


Figure 6: Example of two-level scheduling hierarchy. Group 1, 2, and 3 have the credit of 600, 300, and 900. The sharing ratio between groups is 2:1:3. In each group, CPU time is allocated to individual VMs using their credits (number in each VM) as sharing weight.

Figure 6 from [2]

For Group 1:

CPU allocation for the group: 2/6 of overall CPU time.

Within group sharing ratio: 2:1.

CPU allocation for VMs: 2/9 and 1/9 of overall CPU time.

(2/3).(2/6)=(4/18)=(2/9)

43

Hierarchical Scheduling

- **Possible problem:**
  - It is possible that all the VMs in a group are blocked in the waiting list after requesting too many I/O.
  - → Wasted CPU time.
- **Solution:**
  - A timeout counter is associated with each group.
  - Start the timeout when all VMs within the group are blocked.
  - If the timeout expires and they are still all blocked → the group yields the CPU to the next group.

44

Setting the Timeout Counter

- **Setting it too low:**
  - → The scheduler switches between groups quickly.
  - → Pending I/O are not given a chance to complete.
  - Affects group-level fairness.
- **Setting it too large** → the system is underutilized.
- The scheduler must select it to **balance** these trade-offs.

45

Setting the Timeout Counter

- Credit-remaining ratio (CRR) for each group:

$$CRR = \frac{\sum VM\_remaining\_credit}{Total\_group\_credit}$$

  - The fraction of CPU credits remaining for the group.
  - If CRR is close to 1 → the group is far from using all its credits. While if it is close to zero → the group has used most of its credits.
- t_w = wakeup latency between the time when all VMs were blocked and the time when any of them was boosted. (initialized with 1 ms).
- The timeout: $t = t_w \cdot CRR$

46

Hierarchical Scheduling

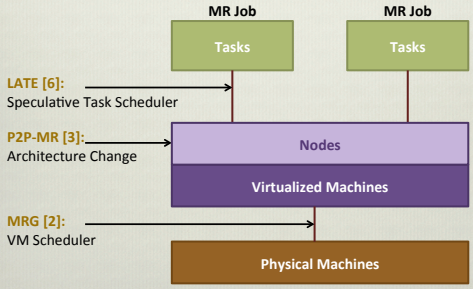
- **Additional extensions:**
  - The current running group is selected as the group of the VM in the head of the running queue.
  - The scheduler must only select VMs from the group.
  - The VMs within a group can exist in different priority regions of the run queue. → The scheduler must check all regions for VMs within the group.

47

Third Approach

48

Three Approaches to Improve the Performance



49

Third Approach

- **P2P–MapReduce, by Marozzo et al. [3].**
  - A peer-to-peer architecture of MR nodes.
  - **Inefficiency** they are addressing: master failure.
  - **Goal:** minimize lost computing time caused by master failure.

50

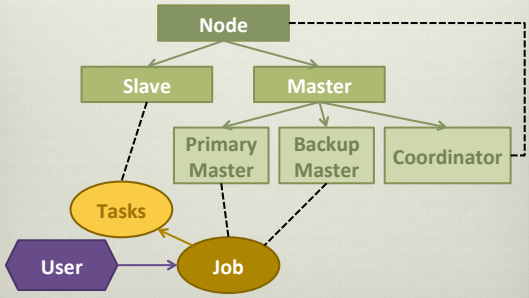
System Architecture

- The system is composed of a set of peer nodes: a number of them are assigned the master role, while the remaining are assigned the slave role.
- The assignment can be changed to maintain the master/slave ratio by the coordinator (only if they are idle).
- Each job has a primary master that keeps the job state (ex: task assignments to slaves, and the status of each task), and a set of backup maters that are updated with a replica of this job state.
- Upon the failure of the primary master, its place is taken by one of its backup masters.

51

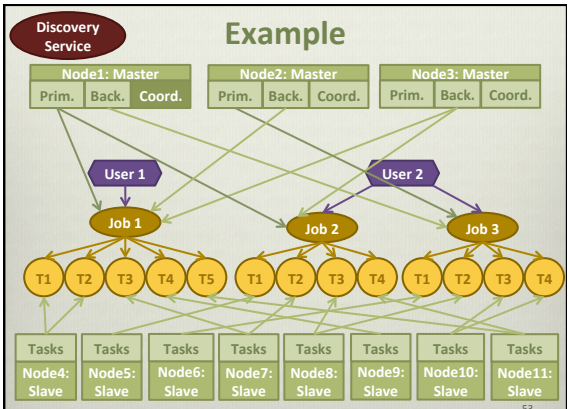
System Model in General

(more details and data structures in the paper)



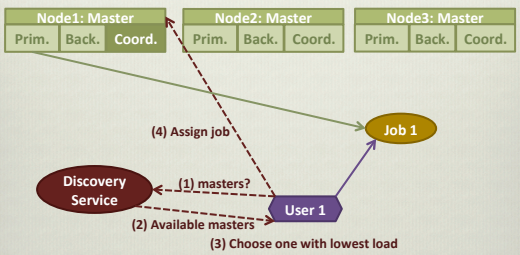
52

Example

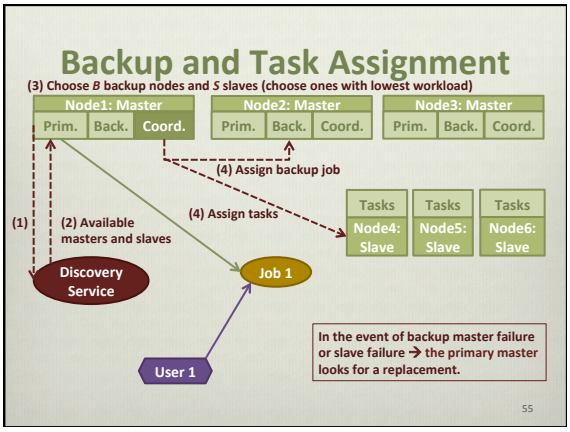


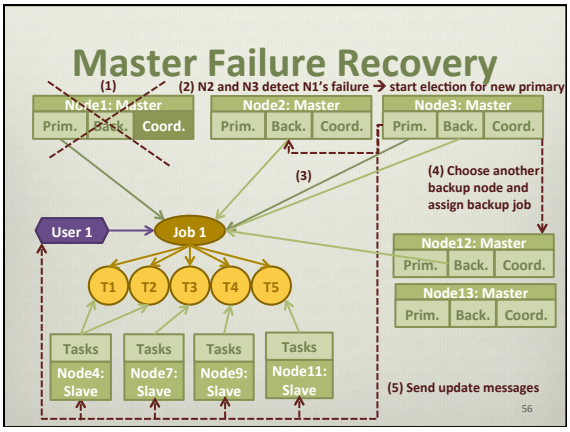
53

Job Submission: Master Choice



54

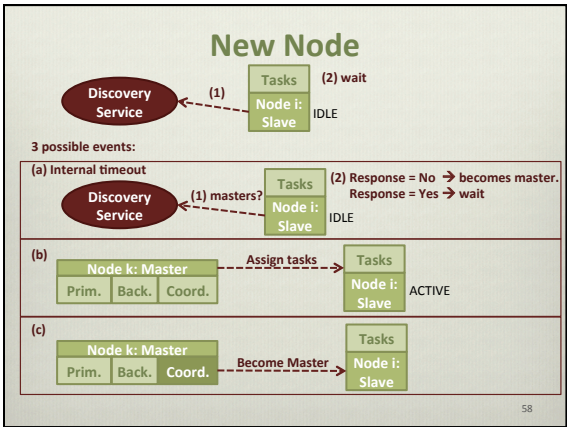




Election Procedure

- For new primary (among backup nodes) or new coordinator (among all nodes):
  - A node sends an election message to all masters with higher ids, then waits for a response.
  - If no response within a time limit → it wins.
  - Otherwise → wait for an update message from the new primary or coordinator.
  - If no update message received before a timeout → restart election.

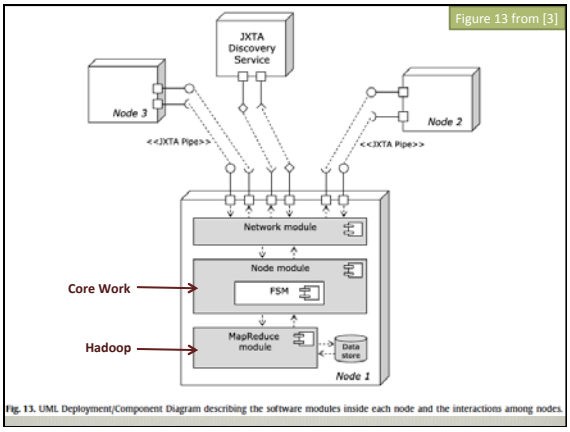
57



Implementation

- P2P-MR was implemented using the JXTA framework.
  - Messages are sent using asynchronous communication pipes.
  - Each node periodically publishes an advertisement with its information (role, workload, expiration time).
  - Each node is composed of 3 software layers.

59



Commonalities, Contrasts, Inefficiencies, and Possible Extensions

61

Commonalities and Contrasts

- **Commonality**
  - Improve MR performance in the clouds environment.
- **Contrasts**
  - The level they operate on.
  - **LATE** requires the usage of the proposed modified MR implementation (**application level**).
  - **P2P-MR** require the usage of the proposed MR middleware (**application level**).
  - **MRG** Scheduler requires a modification in the **OS** of the physical machines (patched Xen). (May be a disadvantage.)

62

Inefficiencies & Possible Extensions

- **LATE Scheduler:**
  - **Target:** a fix in the speculative execution.
  - **Problem:**
    - The authors used two of the assumptions that they invalidated: constant progress rate, and that each phase in a reduce task represents 1/3 of its total time.
    - This caused wrong estimated remaining times for tasks → wrong ranking.

63

Inefficiencies & Possible Extensions

- **LATE Scheduler:**
  - **Possible solutions:**
  - (1) Dropping the the usage of the progress rate in the ranking procedure. (To make it more robust for heterogeneity)
    - Remaining time = (1 – progressScore) / progressRate where Progress rate = progressScore / T
    - Suggested ranking value = T.(1 – progressScore)
    - It is proportional to the amount of progress remaining and how long have the task been running.
    - The older they are and the less progress they made, the more preferable they are for speculative execution.

64

Inefficiencies & Possible Extensions

- **LATE Scheduler:**
  - **Possible solutions:**
  - (2) Improve the progress score calculation. (To adapt to the different jobs requirements: dynamic assignment instead of static assignment)
    - Suggested method: use the user's knowledge about his/her own developed map and reduce functions.
    - When the user submits these codes through the MR interface, the user can also include an estimated time complexity per record read. Then MR uses this information to proportionally divide the scores for the different tasks.

65

Inefficiencies & Possible Extensions

- **MRG Scheduler:**
  - **Problems:**
    - Based on 2 assumptions: MR jobs are I/O intensive, and all nodes in a group have the same map and reduce functions.
      - They may not hold. (ineffective → degradation???)
    - It guarantees proportional fairness to each group.
      - Practical, but not perfect. (Trade-off between fairness, utilization, and context-switch overhead.)
      - Weights play a strong critical role here.
    - May not be fair to other VMs **not** belonging to MR jobs.
  - **Possible Solutions:** ???

66

Inefficiencies & Possible Extensions

- **P2P–MapReduce:**
  - **Problem:**
    - A lot of message passing between the nodes to maintain the system.
      - It gives it the robustness (fault-tolerance), so it is more reliable but at the expense of more work-load and network traffic.
      - The authors preferred management efficiency over network traffic.
  - **Possible Solution:** ???

67

Inefficiencies & Possible Extensions

- **P2P–MapReduce:**
  - **Problem:**
    - Role change only happens for idle nodes.
      - Since load balancing is used to distribute the tasks, there may not exist idle nodes but there may be nodes with very light load.
  - **Possible Solution:**
    - If there is a need to change roles:
      - First, check for idle nodes.
      - If none exists:
        - Look for the node with lightest load.
        - Transfer its load to the second lightest-load node and send the necessary update messages.
        - Change the role of this node.

68

Conclusion

- Three reviewed approaches address the problem of the MR performance degradation in the heterogeneous and dynamic environment of the clouds.
- They all targeted different levels of MR.
- They are not conflicting, but complementing each other.
- It is possible to integrate all the three approaches in a single framework in a future work.

69

Thank you.

70

References (1/2)

1. Dean, J., and Ghemawat, S. **"MapReduce: Simplified Data Processing on Large Clusters"**, Communications of the ACM, Vol. 51, No. 1, 2008, Pages: 107-113.
2. Kang,H., Chen, Y., Wong,, J., and Wu, J. **"Enhancement of Xen's Scheduler for MapReduce Workloads"**, HPDC'11, San Jose, California, USA, 2011, Pages: 251-262.
3. Marozzo, F., Talia, D., and Trunfio, P. **"P2P-MapReduce: Parallel data processing in dynamic Cloud environments"**, Journal of Computer and System Sciences 78, 2012, Pages: 1382–1402.

71

References (2/2)

4. Rehman, M., and Sakr, M. **"Initial Findings for Provisioning Variation in Cloud Computing"**, 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010, Pages: 473-479.
5. Stonebraker, M., Abadi, D., DeWitt, D., Madden, S., Paulson, E., Pavlo, A., and Rasin, A. **"MapReduce and Parallel DBMSs: Friends or Foes?"** , Communications of the ACM, Vol. 53, No. 1, 2010, Pages: 64-71.
6. Zaharia, M., Konwinski, A., Joseph, A., Katz, R., and Stoica, I. **"Improving MapReduce Performance in Heterogeneous Environments"**, 8th USENIX Symposium on Operating Systems Design Implementation, San Diego, CA, 2008, Pages: 29-42.

72