Task Scheduling Strategies for Workflow based Applications in Grids

Many scientific computing tasks are too large to be processed quickly on one computer. Instead, the grid has become popular for a wide range of disciplines. Efficiently using the all the resources of the grid is a challenging, NP problem though. The authors introduced a new method for scheduling that accounted for data dependencies, a topic often ignored in job scheduling algorithms. They also analyzed the method extensively, in terms of tasks that were compute bound, data bound and tasks where job completion was estimated incorrectly.

Scheduling algorithms are required to map a set of jobs to resources at a certain time so that the makespan is minimized (meaning the time from the start of execution until the last job is completed). The naïve approach, called the task based approach, is a greedy local allocation that looks for the shortest estimated time to completion among jobs. The goal is that by reducing the makespan at each step, the overall result will be good. Clearly, it cannot provide optimality though. The workflow based approach takes a more holistic formulation. Here, the entire set of jobs was analyzed at the outset, avoiding the myopia of local decisions. A number of iterations were performed which calculated different schedules, the best being chosen. An element of randomness was introduced in order to find globally optimal allocations more efficiently.

In order to validate their claims, the authors provided extensive simulations built upon NS and using actual astronomical workflows. Two types of flows were investigated, data intensive and compute intensive, illustrating the variation present in scientific datasets. Furthermore, homogeneous and heterogeneous grids were compared. Their results suggested that the workflow based approach did much better for data intensive tasks, since it was able to realize when nodes would become idle due to waiting for files. By pre-positioning data files, large file transfers could begin immediately after creation because they knew where to be sent. This ability came from the use of a new local heuristic selection policy that minimized the idle time in waiting for input files. The weighted min-min heuristic was a way to select the next resource that accounted not only for expected completion time, but also the expected idle time for that job. Further results found that random allocation was sufficient if resources were homogeneous and a uniform job dispersion was needed. As a down side to the workflow based approach, it did not scale well for a large number of jobs, requiring too much CPU time to calculate the next iteration. The last analysis looked at situations where job time was difficult to gauge accurately. In those cases, the authors found that the workflow based approach was better in heterogeneous situations, but performed equally with the task based approach if the grid was uniform.
Analyzing and transforming large sets of scientific data remains a challenge for many disciplines, such as astronomy and biomedical image analysis. One category of data processing can be defined as a set of tasks with batch-shared I/O in which files are shared across tasks. More specifically, data sets are stored in different files on separate storage cluster nodes and subsequently analyzed on compute cluster nodes. Tasks are independent and sequential, requiring a subset of the files. Appropriately scheduling these tasks among different resources is a problem that deserves attention. The authors developed a new method for scheduling based on the idea that many tasks would request the same files. They first partitioned tasks into groups using a hypergraph, then determined the best order of execution on each node. The goal was to minimize the total batch execution time.

Prior approaches have been more generally focused on execution time, such as min-min, max-min and sufferage. The first finds the task that can complete the earliest and assigns it to the node that can execute the fastest. Max-min selects the task with the maximum completion time and sufferage finds the task that will suffer most if it's not scheduled immediately. The authors proposed approach models the sharing of files amongst tasks with a hypergraph. The hypergraph then partitioned the tasks into n-ways so that the cutsize was minimized in the hypergraph (the min cut size is seemingly similar to the min cut in normal graphs). By partitioning into groups, the amount of data transfer was minimized and the load was spread evenly. This is a well studied problem, and although it is in NP, there are optimizations that make it efficient. Note that this grouping does not yet have an order.

The second step in the methodology was to order tasks and schedule the transfer of data from the storage cluster. Data transfer was scheduled so that contention was minimized amongst tasks. A Gantt chart was used to find the scheduling by maintaining time slots for tasks (a Gantt chart is basically a bar chart for scheduling). The duration of each time slot was calculated as the earliest completion time of a task (which was an amalgamation of execution, data transfer and data loading time).

The experimental setup used real data sets with varying degrees of overlap of data usage between tasks. Predictably, for high overlap the hypergraph partitioning did much better than other techniques, but for lower overlap the methods were comparable. Because the hypergraph can cluster tasks that share files together, it significantly reduced data transfer time from remote storage. The takeaway lesson from the paper was that ordering and distributing task load needs to consider file transfer in order to achieve optimality.
Data Cube: A relational Aggregation Operator

A noticeable missing feature of SQL is the ability to have multi-dimensional generalizing aggregators. In terms of visualizing multi-dimensional data, it is imperative to reduce dimensions through summarization for better understanding. Retrieving and presenting this data (and doing so efficiently) is a challenge the authors attempted to resolve. The standard aggregization features include function like sum, min, max, average as well as the group by which partitions data into groups. Other non standard, but available, aggregators are able to provide rank, percentile and ratio, while also allowing user defined functions to be called by the database. The authors identified the main problem with the current standards in that they cannot create histograms, roll up totals and cross tab easily. For example, it would be useful to perform some aggregation within the group by clause. Instead, nested queries must be used to create histograms. Creating roll-ups with totals and sub-totals is even more difficult, since aggregation is done in a nested manner. A number of previous proposals have been offered that allow for representing this data, but each had serious drawbacks, such as exponentially increasing columns. Instead a new ALL keyword was defined which is a convenient way to insert tuples that would otherwise have NULL values. Then by unioning all the various roll-ups, a SQL command could be constructed representing the desired output. Unfortunately, writing these commands, as well as processing them, became a daunting task.

The solution was to employ the use of a data cube in the grouping section. The cube operator took all the attributes of the group by, performed the aggregation and did a union, substituting ALL for the aggregated columns. If there were N attributes, then $2^N$ super aggregate values would be created. Since the full cube was sometimes too much data, a ROLLUP operator was also provided which outputted just the aggregation values.

The ALL operator, although providing a nice abstraction, actually represented a set, meaning it had the potential for opening up a new world where values could actually be relations. Adding new keywords to SQL also presented challenges that added complexity to the overall implementation, such as defining the meaning of binary operations for ALL, making sure it did not participate in count() queries and installing it as a reserved word. Although ALL gave a new way to retrieve data, it was not required, the authors having implemented another command, GROUPING, which could perform the same functionality. Additionally, in extending the group by operator, decorations were introduced so that extraneous labels could be selected for even though they do not affect the grouping.

Performing the cube operation within the database has no simple method. The $2^N$ algorithm was defined in which tuples must be handled $2^N$ times. Other basic techniques for aggregation were performed such as computing them at the lowest level, using arrays and hashing to organize data, and using parallel computation if the set was too large for one disk. Computing super aggregates could take a number of forms depending on the type of operation. For example distributive operations were defined as types like Minimum, which need only scan the data once. Algebraic are more complicated, such as averaging, and holistic operations require the use of the $2^N$ algorithm, such as median and rank.

The final section mentioned an interesting case of using cube operators to maintain triggers. When data was modified, the entire cube would be updated. For some operations, like insert, the cost for maintenance was small, but interestingly on a delete/update, simple operations like finding the minimum had to be completely recomputed, changing the nature of the operation from distributive to holistic.