Steady-State Scheduling, part 2

Loris Marchal

1 Principle of steady-state scheduling

Summary of last lecture : article from Bertsimas & Gamarnik

- packet routing with fixed path
  - fluidified version of the problem (rational numbers instead of integers)
  - optimal fluid solution is easy (formulas for throughputs and buffer sizes)
  - rounding of the fluid solution : periodic schedule, rounding rational numbers to integers
  - length of the period : square root of the (fluid) optimal makespan, best trade-off between
    - long and efficient period (not too much idle time)
    - large number of period, to minimize latency
- packet routing without fixed path
  - fluidified problem : no more simple solution, but easily written as a (rational) linear program
  - rounding of the solution : similar, but much more tricky to bound the number of remaining packets
  - (remaining packets : routed separately !)
  - periodic schedule, same discussion on period length

Principles

- focus on steady-state, forget transient phase
- optimize throughput during central steady-state
- in this article : trade-off between the loss in steady-state, and the loss in initialization and clean-up phases (period length = square root of optimal makespan)
- other solution : get optimal steady-state schedules
- as soon as the number of packets is large, the solution is asymptotically optimal :

$$\frac{C_{\max}}{C_{\max}^{\text{opt}}} \xrightarrow[n\to\infty]{} 1$$

2 Steady-state scheduling for a similar problem

- Let's get a more realistic network model :
  - Given topology (graph)
  - Sending a unit-size message from P_i to P_j takes a time $c_{i,j}$ (edge weight). For a message of size S, it will take $S \times c_{i,j}$. Note that we might have $c_{i,j} \neq c_{j,i}$.

- Each processor can send (and receive) a single message at a time (bidirectional one-port model).
- During a communication of size S from P_i to P_j starting at time t (i.e., during $[t, t + Sc_{i,j}]$:
  - P_i cannot start another sending operation
  - P_j cannot start another reception
  - P_j cannot forward the message, or start a computation depending of this message

We consider here a new problem : Scatter
- scatter : one source processor sends a distinct message to a set of target processors
- series of scatter : similar to scatter big messages using pipelining

Notations for average (fractional) numbers
- $n(P_i \to P_j, k)$: average number of messages of type k (that is, targeting P_k) send through edge (i, j) during one time unit
- $s(P_i \to P_j)$: average occupation time of edge (i, j) during one time unit

Constraints
- one-port : outgoing messages, incoming message
- relation between n and s
- conservation law
- throughput definition

We get a linear program. Note that all valid solution can be described as n and s, and must follow the linear program. Hence the throughput of an optimal solution of the linear program is a lower bound on the achievable throughput.

From a solution of the linear program to a real solution :
- Rational numbers : compute the lowest common multiple (lcm) of all numbers of messages, and multiply all quantities by this number
  - lcm polynomial in the input parameters of the linear program
  - potentially large period, may be shorten using approximate solution
- One-port model : from local constraint to a valid global schedule (example from the JPDC article)
  - graphs of communication (split a node in receiver/sender)
  - one-port model : a valid pattern is a matching in this graph
  - algorithm to decompose the graph in a weighted sum of matching, such that the sum of the weight is no more than the weight of a node in the graph
  - extract matchings to organize communications (if needed, avoid splitting messages by multiplying by lcm again)
- Initialization and clean-up phases :
  - Initialization : the source processors first sends all needed messages to everybody, OR compute the first activation of communications using a graph traversal...
  - Clean-up : similar.

Asymptotic optimality

- Every valid schedule has a throughput lower than ρ^*, throughput of an optimal solution of the linear program
- Let T_i be the time needed for initialization and clean-up (T_i constant in the number of messages send).
- Throughput for a time T : $\frac{T+T_i}{T}\rho^*$
- Asymptotically optimal

Conclusion

- Benefits :
  - Simplicity (description : one period)
  - Efficiency (asymptotic optimality)
  - Adaptability ? (measure bandwidth during one period, change the schedule for the next one
- Drawbacks :
  - Complexity (statically allocate specific path to each packet)
  - Bad performance for small batches
  - Need for large buffers

3 Adding computations : bags of tasks

What if we add some computations :

- independent tasks to be distributed, computed, and results gathered
- a.k.a. master/slave tasking
- similar to divisible load without the divisible assumption (tasks are less numerous)

New notations

- $n_{\text{data}}(i,j)$: average number of data files send through an edge (i,j) during one time unit
- $n_{\text{data}}(i,j)$: average number of result files send through an edge (i,j) during one time unit
- $n_{\text{proc}}(i)$: average number of files processed by node i during one time unit

Constraints

- Similar constraints for one-port model, and to define s
- New conservation laws :

$$\begin{aligned}\sum_j n_{\text{data}}(j,i) &= \sum_j n_{\text{data}}(i,j) + n_{\text{proc}}(i)\\ \sum_j n_{\text{result}}(j,i) + n_{\text{proc}}(i) &= \sum_j n_{\text{result}}(i,j)\end{aligned}$$

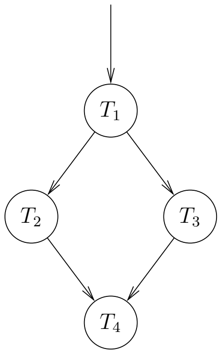
Schedule reconstruction and asymptotic optimality

- Building a schedule : same for communications, nothing to do for computations
- Same proofs for performance

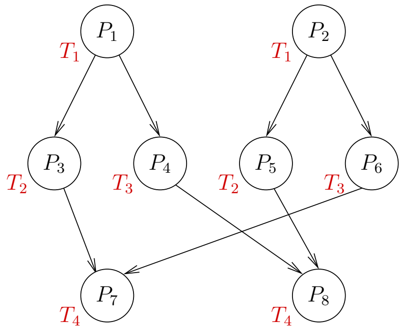
4 Task graphs

Let's consider an even more general problem : task with dependencies (task graphs, DAG). Maybe, we can do the same : add a variable for each task type, and (complex) conservation laws.

Counter-exemple :



Application graph



Platform graph : each processor is able to process only one task type

- We need to precisely reconstruct the data paths for each instance in the flow
- Allocation : set of operations needed to process one instance of the series (computations and communications)
- For the DAG, an allocation is the DAG mapped on the platform graph
- Need to extract the solution as a weighted sum of allocations from a solution of the linear program : not always feasible
  - scatter : an allocation is a set of path from the source to each destination, which is easy to extract
  - DAG : we need to tag each parent task, to know on node it was processed, in order to reconstruct the allocations
  - broadcast : a broadcast tree (feasible, but complex)

A more general formulation :
- communication pattern may be more complex than matchings in a bipartite graph (think of the unidirectional one-port model : matchings in a general graph)
- a solution is a combination of allocations (user point of view)
- a solution is a combination of matchings (resource point of view)
- linear program with one variable per matching and per allocation
- exponential number of variables/polynomial number of constraints
- can be solved using the ellipsoid method, or (maybe) column generation