Big Computing
Parallel Terms

Warwick RSE

December 2022

Letter Sorting Script

- Mail Sorting as a parallel problem
- Suppose you are designing a system to sort letters by address to enable them to be delivered. Assume you have multiple people available to physically carry out the task, and it is up to you to say how they should go about this.
- THIS IS NOT meant to be a direct analogy to parallel methods, but it is going to show how many things you might have encountered in life share some of the issues of parallel programming and can provide insight into the solutions. In other words, you may never have dealt with parallel programming formally, but you have probably already met a lot of the skills involved.
- So, the sorting of letters.
- A very important early step, which is often overlooked, is to make very sure about what the actual aim of your system is going to be. It's all very well to say you're going to "sort" things, but have you considered what level of granularity is needed? What constraints might there be on the piles you are trying to create? Do you want things divided by street? By city? Perhaps you are targeting actually delivering these letters (if so perhaps Royal Mail has something to learn from you) - in which case you probably want letters sorted into groups that form coherent post routes, which is similar to, but also rather different from, the question of simply arranging in order.
- As well as the end goal of your system, you must consider the start parameters. Do you have a large number of letters compared to your number of workers? Is distributing the letters to the workers likely to be a crucial part of the system? For instance, will they have to queue to collect work, or can you physically arrange things to avoid this? Can you randomly hand them out, or are they already somehow divided?
- What about the "running" parameters? Are some workers likely to be faster than others, or might some letters inherently take longer to sort (load imbalances)? Do you prefer workers to get overloaded, or to sit idle - because try as you might, perfectly loaded is rarely the case.

Letter Sorting Script

- Suppose you plan a simple system where each worker takes a stack of letters, and places them in pigeon holes according to the street address. Might you see problems with "resource contention" - people having to queue to receive a new batch of letters, or having to queue to place letters in particularly heavily populated bins?

- How much work do you give to each individual worker at a time? One letter? Or some size of batch? Might the batch size depend on the number of workers, or the number of letters?

- What if you are receiving your letters in a continuous stream, that can sometimes be only a trickle? How long do you wait to get "enough" work to occupy a worker?

- Now consider the actual "sorting" process? For instance, does it require somebody remember which city is associated with a given post code, or some other information which might push you towards a "production line" type of sort, where people handle sub-parts of the sorting task, rather than subsets of the letters?

- Perhaps you might consider a "selection line", passing letters along until they reach the worker responsible for picking out those with Post code CV4, and make your first subdivision that way. How many workers per post code? What about letters that fall off the end of the line?

- Real world letter sorting might even involve some sort of "difficult letters office" - a worker dedicated to reading smudged writing, interpreting approximate addresses and other such interesting issues. Does your system need this kind of fall-back?

- Coming up to Christmas, we might as well think about what we'd do with all the letters to Father Christmas that we'll see, and other special cases like that. We dont want our workers to have to "switch task" to handle these as that interrupts their workflow, so ideally we handle these just like normal letters.

Letter Sorting Script

- In fact these sorts of changes in workflow might be common, and they can often have unintended effects. We might think that flagging certain letters as special, or high priority, will get them handled faster, but in reality, the disruption can exceed the benefit and we have to be very careful about this. Sometimes it can even be better to do redundant work than to let special cases proliferate.

- Now consider what we've seen here. We've thought about the inputs and outputs of our system, the processing involved in the middle, how the processing might inform our approach. We've had to consider the challenges posed by matching workers to work, optimising how many letters we are able to handle. We've seen problems such as workload imbalances, contention for resources, problems with workers requiring context to do their job, problems with special cases. We've seen that there are several different approaches we might take, and that these have their positive and negative features - the "right" approach depends heavily on how we want to trade off problems against each other. Some solutions require detailled procedures, others simply set workers going and wait. Sometimes the right approach is counter-intuitive and doing something a long, systematic way turns out to work better on average than a series of special cases.

Load Imbalance

- Tasks take different amounts of time to complete
  - Might depend on input parameters
  - Might depend on external factors
- General problem referred to as suffering "*load imbalance*" between tasks
- Solving this involves "*load balancing*"

Resource Contention

- Resources might be limited to one accessor at a time
  - E.g. writing to a file (mostly)
- If two tasks independently try and access this, it's called *"resource contention"*
  - Obvious solution; make one wait until the other finishes. But:
    - This part of job is not now parallel (has been *serialised)*
    - If A waits for B, but B is waiting for something else from A, nobody moves (*deadlocking*, see later)

Task Switching

- Suppose a worker is doing one task and is then asked to switch and do another
  - Takes time (overhead) to redirect attention
    - E.g. don’t react to emails as they come in or you’ll get nothing done…
- Computers suffer this too
  - Have to store current data, load new stuff etc
  - Can be inefficient and even risky