Parallel Download Optimization

Jan Palášek

2020-11-15

Motivation

We want to download a file of size s. We can download it parallely in away, that we split the file into many pieces and download those pieces separatelly at the same time. Unfortunately, the conditions have changed and we can start a new downloading thread only after m seconds. We also have an internet with a limited speed v_l. What would be the n ideal number of threads to use? The second question is, given $n$number of threads, what would be the ideal length of parts downloaded parts $s_1, ..., s_n$?

Derivation

The main idea is that the downloading is fastest when the most threads possible are running in parallel. Therefore our new method will calculate the threads sizes in a following way: - We start a new thread as soon as possible (after m seconds), - all threads stop running at the same time

This is the optimum way, since there will be running as many threads as possible for as long time as possible.

Denote t_i as the time needed to download part s_i with speed v. Therefore $t_i = \frac{s_i}{v}$ and

$$s = s_1 + ... + s_n$$
$$s = v(t_1 + ... + t_n)$$

We have a limit for the speed: n threads together must not have greater speed than v_l.

$$\sum_i^n v = n \cdot v \leq v_l \rightarrow n \leq \frac{v_l}{v}$$

Since we start a new thread after m seconds and stop all threads at the same time, then $t_i = t_{i-1} - m$. Expanding a few members of the recurrent sequence, we can observe that

$$t_i = t_1 - m(i-1)$$

We can calculate the total amount of time is the time until the last thread stops downloading. The first thread is downloading for t_1 seconds. the second for t_2 and waits m seconds before downloading, therefore the total waiting for the second thread to finish up is $t_2 + m$ seconds. We can expand this up to n.

Thus the total amount of time of the download is

$$t = t_1 = t_2 + m = ... = t_n + m(n-1)$$

The best way is to split it into as many pieces as possible while $\forall i : t_i \geq 0$ holds. Since $t_i < t_{i-1}$, then the constraint $t_n \geq 0$ is the most restricting one. This allows us to ignore the other, less-restricting constraints.

$$t_n = t_1 - m(n-1) \geq 0 \rightarrow t_1 \geq m(n-1)$$

$$s = v(t_1 + ... + t_n) = v\sum_i^n t_i$$

$$s = v\sum_i^n (t_1 - m(i-1))$$

$$s = v(n \cdot t_1 - m\sum_i^n (i-1))$$

$$s = v(n \cdot t_1 - \frac{m}{2}n(n-1))$$

Using the inequality $t_1 \geq m(n-1)$, we can furthermore derive:

$$s \geq v(n \cdot m(n-1) - \frac{m}{2}n(n-1)) = v(\frac{m}{2}n(n-1))$$

$$\frac{2 \cdot s}{m \cdot v} \geq n(n-1)$$

$$n^2 - n - \frac{2 \cdot s}{m \cdot v} \leq 0$$

$$n_{1,2} = \frac{1 \pm \sqrt{1 - 4 \cdot (-1) \cdot \frac{2 \cdot s}{m \cdot v}}}{2} = \frac{1 \pm \sqrt{1 + 8\frac{s}{m \cdot v}}}{2} \leq 0$$

There must also hold that $n \leq \frac{v_l}{v}$. We want to choose $n \in \mathbb{N}$ that has the biggest value. With the previously stated bounds in mind, we can calculate the result number of threads as

$$n = \text{floor}\left(\min\left(\frac{1 + \sqrt{1 + 8\frac{s}{m \cdot v}}}{2}, \frac{v_l}{v} \right) \right)$$

Let's answer the second question: having n, what would be the ideal splits $s_1, ..., s_n$. Using

$$t_1 = \frac{s}{v \cdot n} + \frac{m}{2}(n-1)$$

we can calculate $s_1 = v \cdot t_1$. Since we can calculate it any t_i using

$$t_i = t_1 - m(i-1)$$

, we can calculate s_i for all i.

Example

Suppose we want to calculate optimal number of threads and later corresponding splits for the following parameters.

$$s = 30MB = 30000kB$$
$$v = 100kB/s$$
$$v_l = 20MB/s = 20000kB/s$$
$$m = 60s$$

What is the best n to choose?

$$\frac{1 + \sqrt{1 + 8\frac{30000}{60 \cdot 100}}}{2} = 3.7$$
$$\frac{20000}{100} = 200$$

Therefore the final optimal number of threads to choose $n = 3$.

Using this n we can calculate size of the splits.

$$t_1 = \frac{30000}{100 \cdot 3} + \frac{60}{2}(3-1) = 160s$$

$$s_1 = v \cdot t_1 = 100 \cdot 160 = 16000kB = 16MB$$

$$t_2 = t_1 - m = 160 - 60 = 100s$$

$$s_2 = v \cdot t_2 = 100 \cdot 100 = 10000kB = 10MB$$

$$t_3 = t_2 - m = 100 - 60 = 40s$$

$$s_3 = v \cdot t_3 = 100 \cdot 40 = 4000kB = 4MB$$

We can at last check, that $s = s_1 + s_2 + s_3$, which we can see that it indeed holds.

The best way to split the files is to let the first thread download 16 MB, the second 10 MB and the last thread 4 MB for a total time of 160 s.

Comparison to a naive method

Let's compare it to a naive method of splitting. The naive method splits the file into n same parts. Therefore

$$s_1 = ... = s_n \rightarrow t_1 = ... = t_n$$

and the total length s is

$$s = \sum_{i}^{n} s_i = n \cdot s_1$$

The total time spent by downloading t is calculated as the time of the initial thread t_1 + m delay for starting every other thread.

$$t^{\text{naive}} = t_1^{\text{naive}} + m(n-1)$$

t^{naive} can be calculated as

$$t_1^{\text{naive}} = \frac{s_1}{v} = \frac{s}{n \cdot v}$$

and thus

$$t^{\text{naive}} = \frac{s}{n \cdot v} + m(n-1)$$

Our method had

$$t^{\text{our}} = t_1^{\text{our}} = \frac{s}{n \cdot v} + \frac{m}{2}(n-1)$$

If we compare them, then

$$\Delta t = t^{\text{naive}} - t^{\text{our}} = \frac{m}{2}(n-1)$$

So if we would start the downloading using both methods at the same time using the same number of threads, then our method would end $\frac{m}{2}(n-1)$ seconds before the naive method.