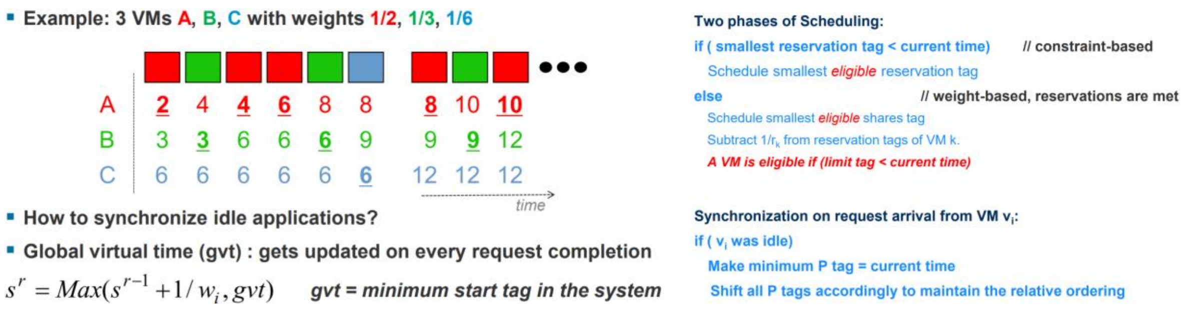
mClock: Handling Throughput Variability for Hypervisor IO Scheduling, OSDI 2010

Many existing fair-queueing techniques have been proposed to support proportional share only, guarantee I/O latency of latency sensitive applications, or guarantee minimum bandwidth of applications.
This paper proposed mClock that meets all of these requirements and easy to implement. mClock is based on the existing virtual time based scheduling, a proportional share scheduling technique.



- Example: 3 VMs A, B, C with weights 1/2, 1/3, 1/6

	🟥	🟩	🟥	🟥	🟩	🟦	🟥	🟩	🟥
A	**2**	4	**4**	**6**	8	8	**8**	10	**10**
B	3	**3**	6	6	**6**	9	9	**9**	12
C	6	6	6	6	6	**6**	12	12	12

- How to synchronize idle applications?
- Global virtual time (gvt) : gets updated on every request completion

$s^r = Max(s^{r-1} + 1/w_i, gvt)$ ***gvt = minimum start tag in the system***

```
Two phases of Scheduling:
if ( smallest reservation tag < current time)        // constraint-based
    Schedule smallest eligible reservation tag
else                                   // weight-based, reservations are met
    Schedule smallest eligible shares tag
    Subtract $1/r_k$ from reservation tags of VM k.
    A VM is eligible if (limit tag < current time)

Synchronization on request arrival from VM $v_i$:
if ( $v_i$ was idle)
    Make minimum P tag = current time
    Shift all P tags accordingly to maintain the relative ordering
```

However, mClock uses real-time tags to guarantee minimum usage and maximum usage of I / O per unit time of each flow.
Unlike the existing virtual time scheduler, which uses only one tag to match the rate of resource usage between each flow, mClock uses two more tags to ensure minimum resource usage and limit maximum resource usage. A total of three tags are used.
mClock select the tag to use by comparing the current time and the tags. and using the selected I/O to dispatch. By doing this, it satisfies all three conditions: minimum resource guarantee, proportional resource usage, and maximum resource usage limit. The algorithm is shown in the upper right corner.

Pros

Unlike conventional I / O schedulers, not only the ratio between flows is considered, but the minimum and maximum consumption can be limited. Through maximum and minimum limit of resource usage, it brings high performance isolation in VM environment.

Cons and Ideas

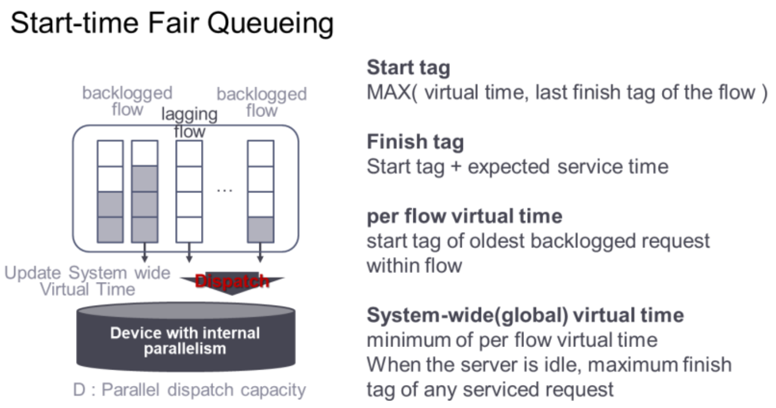
Dispatches I/O with the minimum tags entails lookup of tags of VMs on every I/O dispatch. it would raise scalability issues. And this paper does not serve experiment out of VMs, it would be better to add per task or per application implementations.

FlashFQ: A Fair Queueing I/O Scheduler for Flash-Based SSDs, ATC 2013

In the conventional HDD that handles I/O using a single head, a time slice base scheduler is used to exploit spatial locality. However, this time slice base scheduler has a low responsiveness.

The advent of flash-based devices requires a new I / O scheduler that fits its characteristics. This paper propose a scheduler based on Virtual Time that takes into account the characteristics of the Flash device such as parallelism and reduced importance of spatial locality.

FlashFQ tries to match the fairness between I/Os based on virtual time fair queueing. (specifically, start time fair queueing – I draw SFQ(D) as I understood).


Start-time Fair Queueing
backlogged flow
lagging flow
backlogged flow
...
Update System wide Virtual Time
Dispatch
Device with internal parallelism
D : Parallel dispatch capacity

Start tag
MAX(virtual time, last finish tag of the flow)

Finish tag
Start tag + expected service time

per flow virtual time
start tag of oldest backlogged request within flow

System-wide(global) virtual time
minimum of per flow virtual time
When the server is idle, maximum finish tag of any serviced request

They allow fine-grained interleaving of requests from multiple tasks / flows as long as fair resource utilization is maintained through balanced virtual time progression.

As virtual time fair-queuing is work-conserving scheduler, it does not keep unused resource for inactive task. Therefore, when a task shows deceptive idleness (shortly inactivated), it causes poor fairness. FlashFQ solve this problem by letting a task stay "active" continuously when deceptive idleness appears between its consecutive requests.

Pros

This paper propose a scheduler based on Virtual Time that takes into account the characteristics of the Flash device such as parallelism and reduced spatial locality.

Both fast responsiveness and fairness are satisfied by using characteristics of flash such as parallelism and reduced spatial locality.

Cons & Idea

As virtual time based fair queueing (ex. SFQ) should dispatch I/O from a flow with the minimum virtual time, it should have scalability issues on finding min-value on every dispatch. It would be better to use scalable tree structure for finding minimum virtual time on each dispatch.