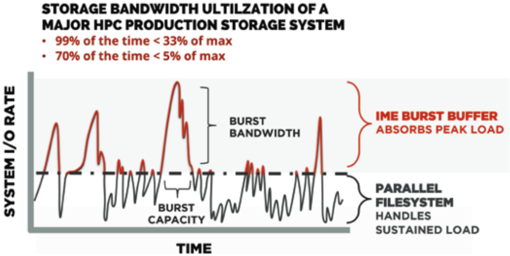
# Sizing and Partitioning Strategies for Burst-Buffers to Reduce Contention

Guillaume Aupy, Olivier Beaumont, Lionel Eyraud-Dubois

# Motivation

**IO congestion in HPC systems:**

- HPC applications are generating lots of data for PFS.
- Idea is to use a buffer when the I/O bandwidth is fully occupied
- The buffer can be emptied at a later time.




Burst-buffers to absorb IO peaks

Source: DDN ad material.

# Application Context

Main transfer source in large HPC applications: checkpoints ($\Rightarrow$ predictable)

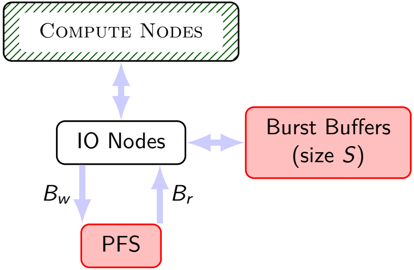
Possible usages for Burst-buffers:

- accelerate one application by caching writes
- hide contention coming from several applications writing at the same time

And with BigData applications coming:

- prefetch input data
- cache for temporary data

# Platform model




Applications run on Compute Nodes

- Placement already done

Two buffer management policies:

- STATIC: size $S_k$ alloted to application $\mathcal{A}_k$ for its lifetime
- DYNAMIC: size devoted to $\mathcal{A}_k$ can change over time

# Application model

Set of applications $\mathcal{A}_k$ running independently on the platform:

- with release date $r_k$, read and write bandwidth $b_k^r$ and $b_k^w$
- consisting of $n_k$ *phases* (without overlap):
  - Read a volume of $R_i^k$ input data starts at $t_{i,k}^r$
  - Compute for $I_i^k$ amount of time starts at $t_{i,k}^c$
  - Write a volume of $W_i^k$ output data starts at $t_{i,k}^w$
- No overlap: data available from the start, but two phases do not fit in memory

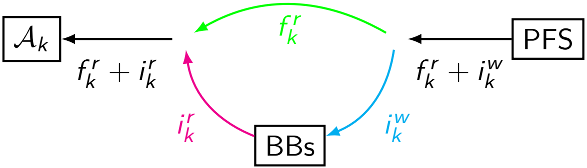
Earliest possible completion time:

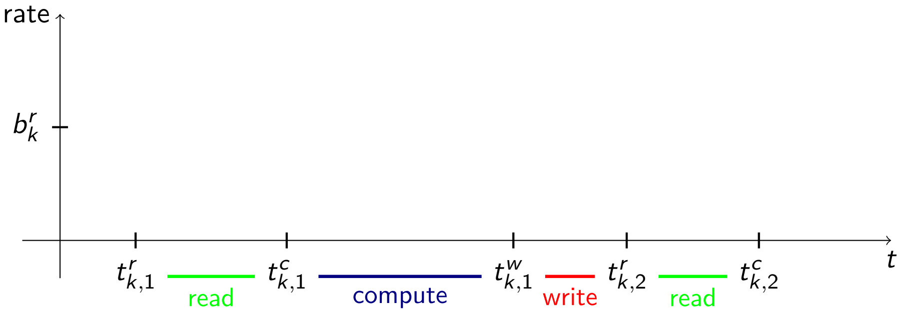
$$C_k^{\min} = r_k + \sum_{i=1}^{n_k} \left( \frac{R_i^k}{b_k^r} + I_i^k + \frac{W_i^k}{b_k^w} \right)$$

Stretch of $\mathcal{A}_k$:

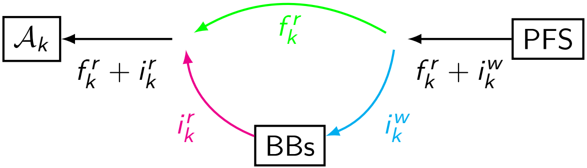
$$s(\mathcal{A}_k) = \frac{C_k}{C_k^{\min}}$$

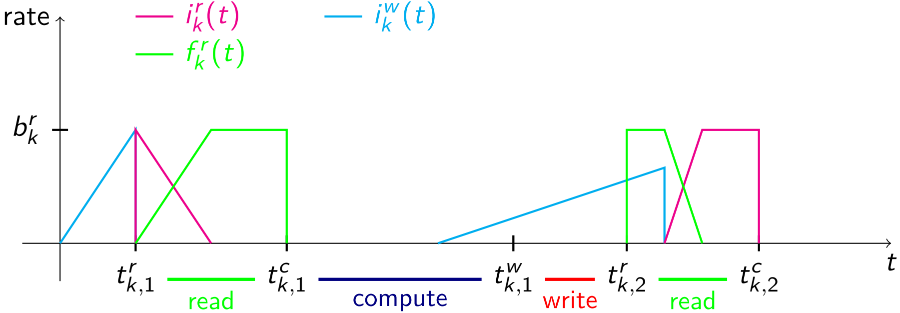
# Execution model



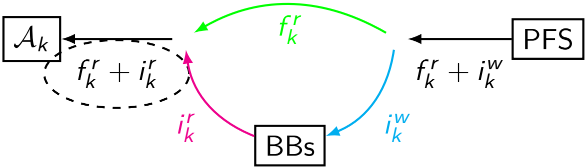



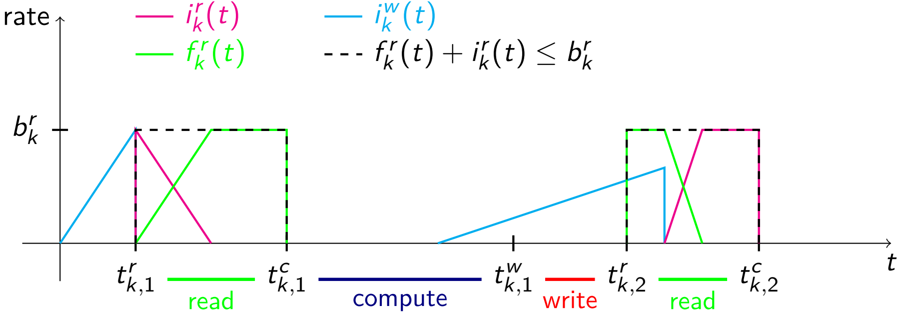
# Execution model



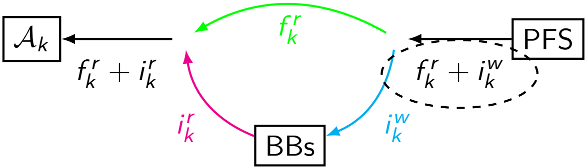



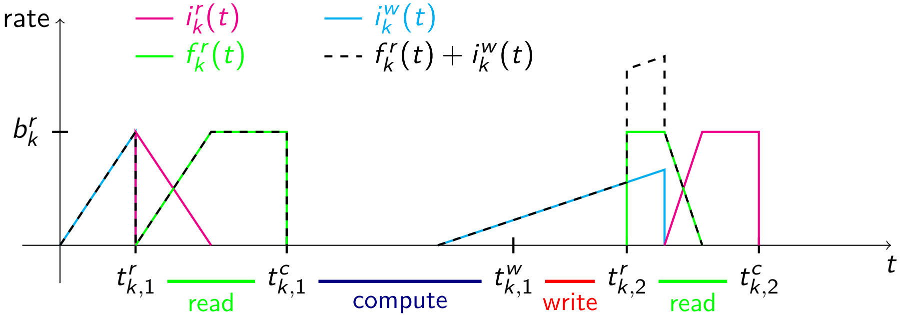
# Execution model



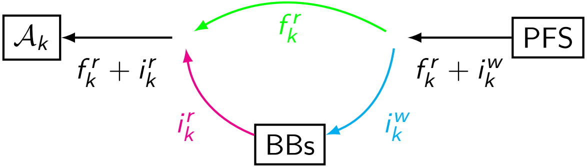



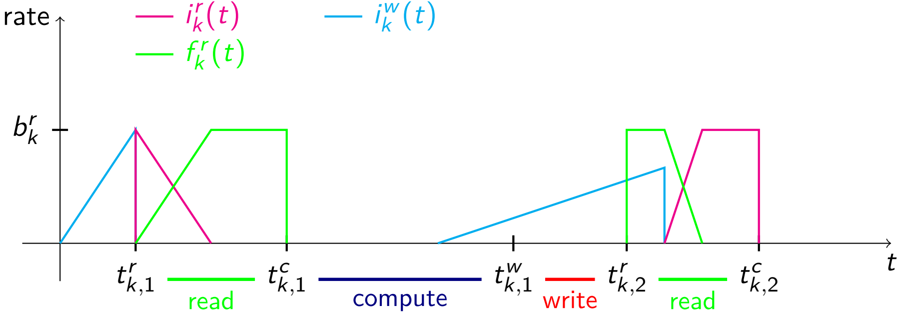
# Execution model



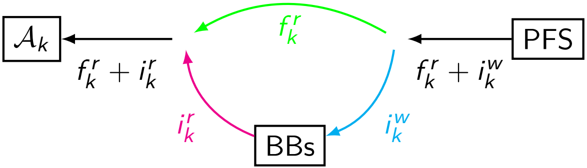
# Execution model

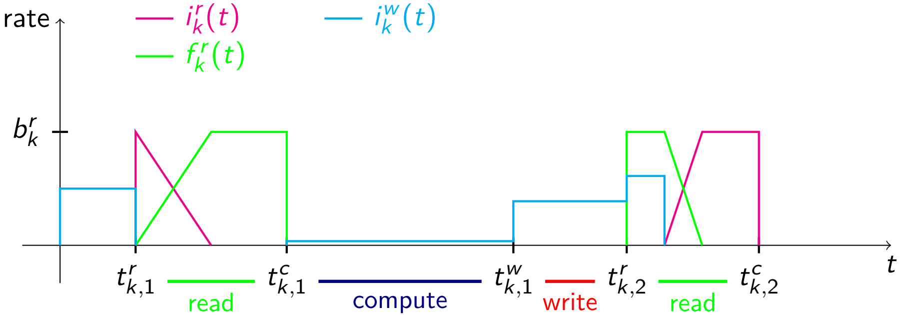



Dominant schedules: all transfer rates *constant* between time events

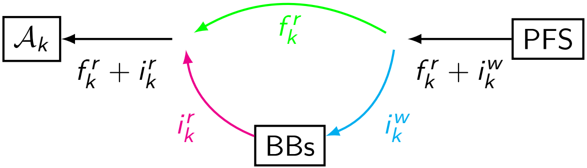
# Execution model

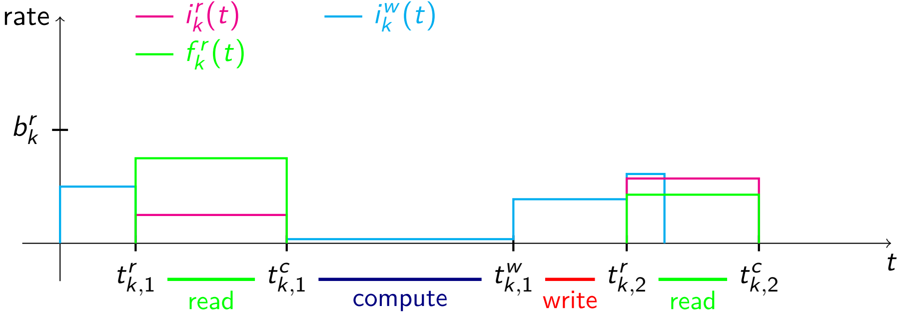



Dominant schedules: all transfer rates *constant* between time events

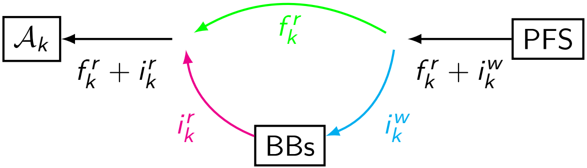
# Execution model

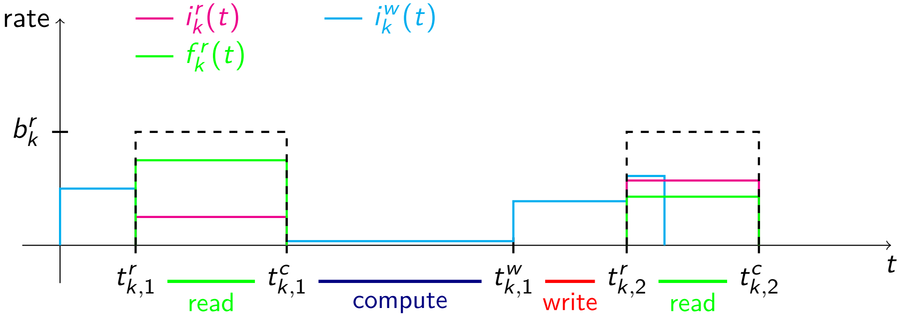



Dominant schedules: all transfer rates *constant* between time events

# Execution model




Dominant schedules: all transfer rates *constant* between time events

Completely determined by the *amount* of data transferred at each event

# Problem formulations – both Static and Dynamic

## Static-Buffer-Size($\rho$) and Dynamic-Buffer-Size($\rho$)

Given $n$ applications $(\mathcal{A}_k)$, and a stretch limit $\rho$, minimize the total buffer size $S$ necessary to achieve stretch $\rho$.

## Static-Stretch($S$) and Dynamic-Stretch($S$)

Given $n$ applications $(\mathcal{A}_k)$, and a buffer size $S$, minimize the maximum stretch over all applications

## Results

- X-Stretch(0) is NP-complete
- X-Buffer-Size($\rho$) is NP-complete for $1 < \rho \leq 2$
- Static-Stretch($S$) is NP-complete for all $S$

# Problem formulations – both Static and Dynamic

### $\textsc{Static-Buffer-Size}(\rho)$ and $\textsc{Dynamic-Buffer-Size}(\rho)$

Given $n$ applications $(\mathcal{A}_k)$, and a stretch limit $\rho$, minimize the total buffer size $S$ necessary to achieve stretch $\rho$.

### $\textsc{Static-Stretch}(S)$ and $\textsc{Dynamic-Stretch}(S)$

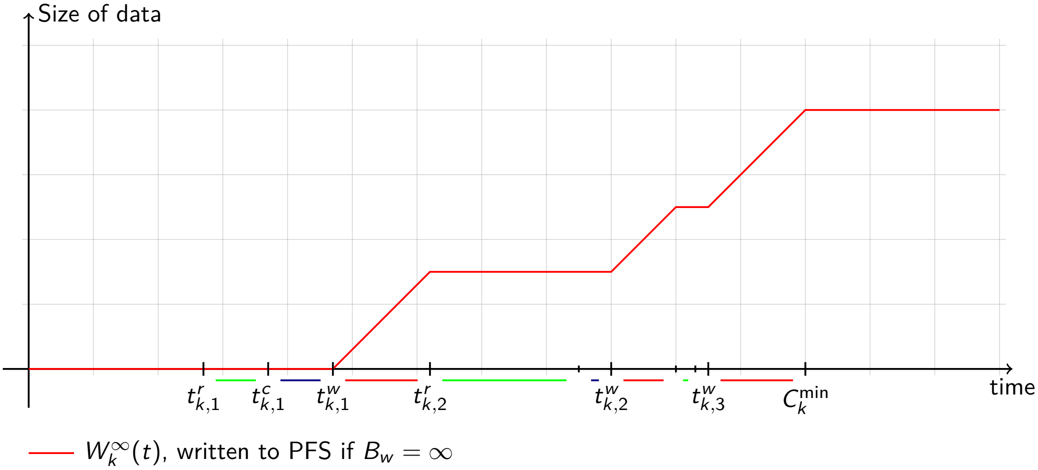
Given $n$ applications $(\mathcal{A}_k)$, and a buffer size $S$, minimize the maximum stretch over all applications

### Results

- $\textsc{X-Stretch}(0)$ is NP-complete
- $\textsc{X-Buffer-Size}(\rho)$ is NP-complete for $1 < \rho \leq 2$
- $\textsc{Static-Stretch}(S)$ is NP-complete for all $S$
- $\textsc{X-Buffer-Size}(1)$ can be solved in polynomial time

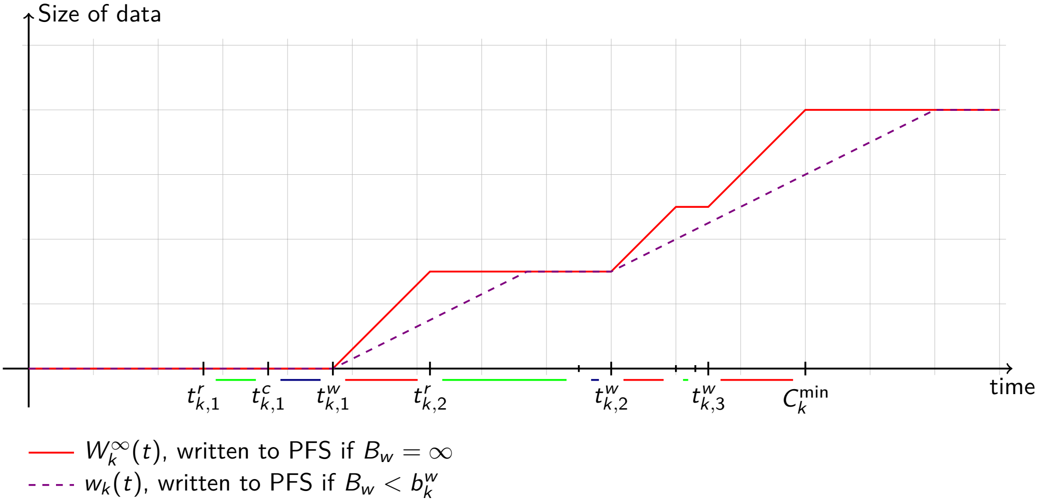
# Scheduling a single application

Aiming for stretch 1 ($C_k = C_k^{\min}$) fixes the values of $t_{i,k}^*$



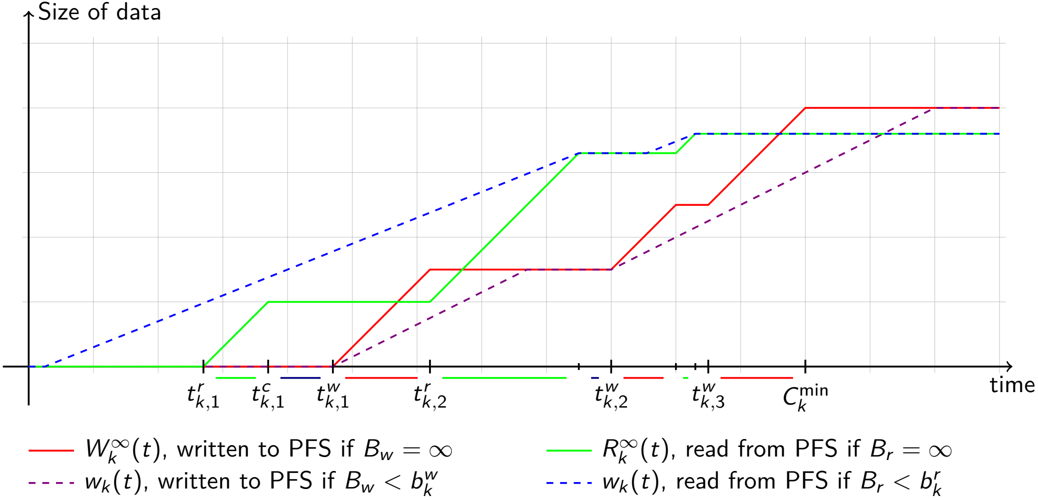
# Scheduling a single application

Aiming for stretch 1 ($C_k = C_k^{\min}$) fixes the values of $t^*_{i,k}$



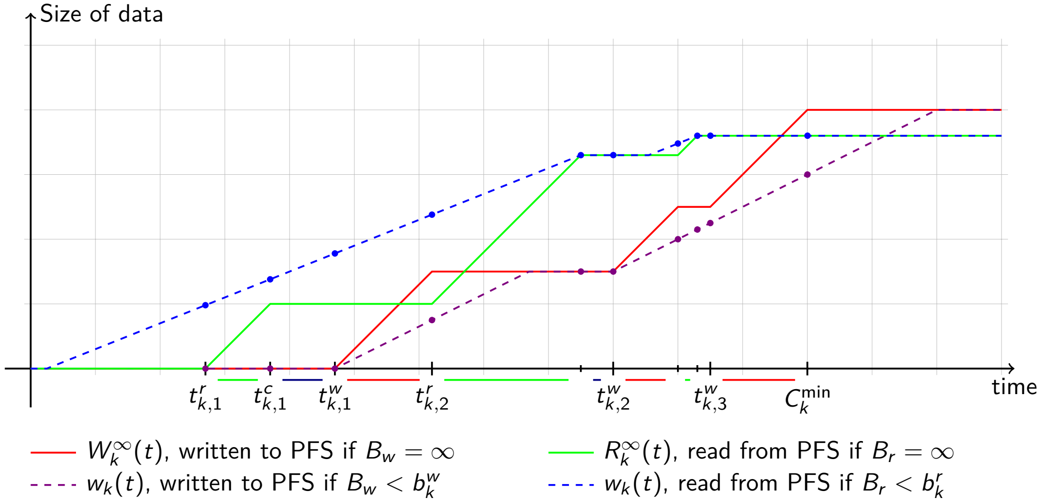
# Scheduling a single application

Aiming for stretch 1 ($C_k = C_k^{\min}$) fixes the values of $t^*_{i,k}$



# Scheduling a single application

Aiming for stretch 1 ($C_k = C_k^{\min}$) fixes the values of $t^*_{i,k}$



# Linear Programming Formulation

Consider all events $\{e_l\} = \{t^*_{k,i}\}$: variables $w_k^l$ and $r_k^l$, variable $S$ for buffer size

Minimize $S$ subject to:

$$\forall l, \quad \underbrace{W_k^\infty(e_l) - w_k^l}_{\text{output data}} + \underbrace{r_k^l - R_k^\infty(e_l)}_{\text{input data}} \leq S \qquad \text{Data stored in buffer}$$

$$\forall l, \quad w_k^l \leq w_k^{l+1} \qquad \text{Amount of data is non-decreasing}$$

$$\forall l, \quad w_k^l \leq W_k^\infty(e_l) \qquad \text{Can not write more than app. sends}$$

$$\forall l, \quad w_k^{l+1} - w_k^l \leq B_w(e_{l+1} - e_l) \qquad \text{Can not write faster than PFS accepts}$$

$$\forall l, \quad r_k^l \leq r_k^{l+1} \qquad \text{Amount of data is non-decreasing}$$

$$\forall l, \quad r_k^l \geq R_k^\infty(e_l) \qquad \text{Must read at least what app. needs}$$

$$\forall l, \quad r_k^{l+1} - r_k^l \leq B_r(e_{l+1} - e_l) \qquad \text{Can not read faster than PFS provides}$$

# Linear Programming Formulation

Consider all events $\{e_l\} = \{t^*_{k,i}\}$: variables $w^l_k$ and $r^l_k$, variable $S$ for buffer size

Minimize $S$ subject to:

| | | |
|---|---|---|
| $\forall l,$ | $\underbrace{W^\infty_k(e_l) - w^l_k}_{\text{output data}} + \underbrace{r^l_k - R^\infty_k(e_l)}_{\text{input data}} \leq S$ | Data stored in buffer |
| $\forall l,$ | $w^l_k \leq w^{l+1}_k$ | Amount of data is non-decreasing |
| $\forall l,$ | $w^l_k \leq W^\infty_k(e_l)$ | Can not write more than app. sends |
| $\forall l,$ | $w^{l+1}_k - w^l_k \leq B_w(e_{l+1} - e_l)$ | Can not write faster than PFS accepts |
| $\forall l,$ | $r^l_k \leq r^{l+1}_k$ | Amount of data is non-decreasing |
| $\forall l,$ | $r^l_k \geq R^\infty_k(e_l)$ | Must read at least what app. needs |
| $\forall l,$ | $r^{l+1}_k - r^l_k \leq B_r(e_{l+1} - e_l)$ | Can not read faster than PFS provides |
| $\forall l,$ | $w^{l+1}_k - w^l_k + r^{l+1}_k - r^l_k \leq B(e_{l+1} - e_k)$ | Total bandwidth not exceeded |

# For several applications

Compute all events $\{e_l\} = \{t^*_{k,i}\}$ for all applications $\mathcal{A}_k$, in increasing order

Variables: $w_k^l$ and $r_k^l$, $S_k^l$ (buffer size of $\mathcal{A}_k$ at event $e_l$)

Minimize $S$ subject to:

| | | |
|---|---|---|
| $\forall l,$ | $\sum_k S_l^k \leq S$ | Total buffer size |
| $\forall l, k$ | $W_k^\infty(e_l) - w_k^l + r_k^l - R_k^\infty(e_l) \leq S_k^l$ | Data stored in buffer |
| $\forall k, l \in \mathcal{I}_k$ | $S_k^l = S_k$ | STATIC constraint |
| $\forall l, k$ | $w_k^l \leq w_k^{l+1}$ | Amount of data is non-decreasing |
| $\forall l, k$ | $w_k^l \leq W_k^\infty(e_l)$ | Can not write more than app. sends |
| $\forall l,$ | $\sum_k w_k^{l+1} - w_k^l \leq B_w(e_{l+1} - e_l)$ | Can not write faster than PFS accepts |
| | $\cdots\cdots$ | [Read constraints] |

# Settings: applications from LANL Computing Center

| Workflow | EAP | LAP | Silverton | VPIC |
|---|---|---|---|---|
| Frequency | 65 | 21 | 8 | 6 |
| # cores (1000) | 16 | 4 | 32 | 30 |
| Ckpt size (GB) | 3,200 | 2,000 | 44,800 | 3,750 |
| Walltime (hours) | 16 | 4 | 32 | 30 |

Platform characteristics:
$96,000$ cores
$B = 160\text{GB/s}$
$b = 20\text{MB/s}$ per core

Period $P = \sqrt{2C\frac{\mu}{\#\text{nodes}}}$
5 years $\leq \mu \leq$ 50 years

## Generating instances

- Fix load $\rho$ to 20, 50 or 80%
- Pick 30 applications according to *Frequency*
- Scheduling them FIFO yields release times
- Compute maximum average bandwidth requirement
- Scale checkpoint size to achieve max. load $\rho$

# Settings

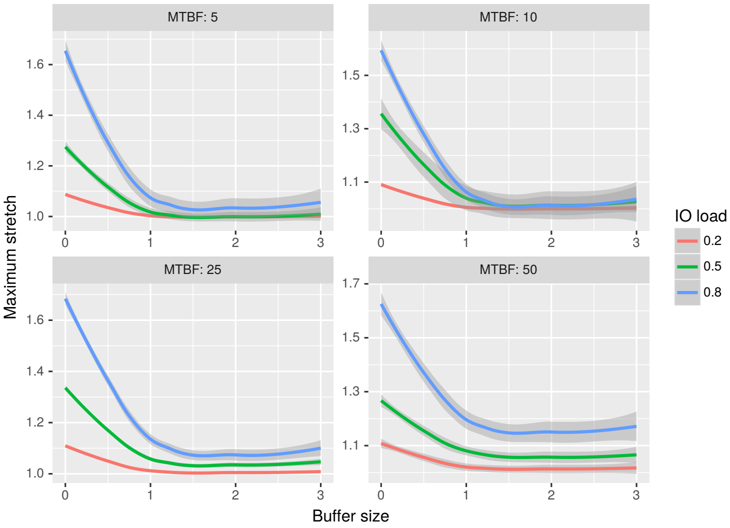
## Greedy strategy

- $\mathcal{A}_k$ is *active* if it is in a write-phase or has data in its buffer
- Bandwidth to the PFS is equally shared between all active applications
- $\mathcal{A}_k$ sends data at maximum rate $b_k^w$ if its buffer has available space
- Otherwise, $\mathcal{A}_k$ is limited to the available bandwidth to PFS

## Methodology

- For each instance, compute $S^{\mathrm{OPT}}$ and $S_k^{\mathrm{OPT}}$ with $\textsc{Static-Buffer-Size}(1)$
- $S$ varies between 0 and $3S^{\mathrm{OPT}}$, scale $S_k$ accordingly
- Apply Greedy strategy, compute maximum stretch

# Results



MTBF: 5
MTBF: 10
MTBF: 25
MTBF: 50
1.6
1.4
1.2
1.0
1.5
1.3
1.1
1.7
0
1
2
3
Maximum stretch
Buffer size
IO load
0.2
0.5
0.8

# Overhead of STATIC vs DYNAMIC

Compare the optimal solutions with and without the STATIC constraint

| Load \ MTBF | 5 y | 10 y | 25 y | 50 y |
|---|---|---|---|---|
| **20%** | 1.32 | 1.31 | 1.42 | 1.67 |
| **50%** | 1.33 | 1.28 | 1.26 | 1.47 |
| **80%** | 1.23 | 1.26 | 1.25 | 1.35 |

Static constraint yields roughly 30% overhead

# Conclusions

## Remarks about model

- Previous paper [IPDPS'18]: random application behavior, Markov Chain modeling
- Here: all phases of applications are known, Linear Programming optimal solution

## Perspectives

- Within this model: efficient strategies for small buffer size
- Extend the model: data reuse, temporary checkpoint data