Being Efficient in Time, Space, and Workload: a Self-stabilizing Unison and its Consequences

S. Devismes, D. Ilcinkas, C. Johnen and **F. Mazoit**

Université de Picardie Jules Verne and **Université de Bordeaux**, France

Synchronous vs. Asynchronous

Many kinds of **“daemons”**

Synchronous vs. Asynchronous

Many kinds of **“daemons”**

Synchronous daemon

At each step, **all** nodes are synchronously activated.

Synchronous vs. Asynchronous

Many kinds of **“daemons”**

Synchronous daemon

At each step, **all** nodes are synchronously activated.

Distributed fair daemon

At each step, **some** nodes are activated.

Synchronous vs. Asynchronous

Many kinds of **“daemons”**

Synchronous daemon

At each step, **all** nodes are synchronously activated.

Distributed fair daemon

At each step, **some** nodes are activated.

No contraints other that nodes **cannot** “starve”.

Synchronous vs. Asynchronous

Many kinds of **“daemons”**

Synchronous daemon

At each step, **all** nodes are synchronously activated.

Distributed fair daemon

At each step, **some** nodes are activated.

No contraints other that nodes **cannot** “starve”.

Distributed unfair daemon

At each step, **some** nodes are activated.

Synchronous vs. Asynchronous

Many kinds of **"daemons"**

Synchronous daemon

At each step, **all** nodes are synchronously activated.

Distributed fair daemon

At each step, **some** nodes are activated.

No contraints other that nodes **cannot** "starve".

Distributed unfair daemon

At each step, **some** nodes are activated.

No constraints. Thus nodes **can** "starve".

Self-Stabilization

The initial configuration is **arbitrary**.

Models error recovery after “transient” faults.[1]

[1] TCP is a self-stabilizing heuristic.

Self-Stabilization & Unison

Self-Stabilization

The initial configuration is **arbitrary**.

Models error recovery after "transient" faults.[1]

(Asynchronous) Unison

Each node has a **local clock**.
Neighboring clocks differ by ≤ 1 increment.
All clocks increase infinitely often.

[1] TCP is a self-stabilizing heuristic.

Self-Stabilization & Unison

Self-Stabilization

The initial configuration is **arbitrary**.

Models error recovery after “transient” faults.[1]

(Asynchronous) Unison

Each node has a **local clock**.
Neighboring clocks differ by ≤ 1 increment.
All clocks increase infinitely often.

A consequence

Run a self-stabilizing algorithm under a synchronous daemon.

[1] TCP is a self-stabilizing heuristic.

Summary so far

Being Efficient in **Time, Space**, and **Workload** :

a **Self-stabilizing Unison** and its **Consequences**

Summary so far

Being Efficient in Time, Space, and Workload : a Self-stabilizing Unison and its Consequences

Complexity measures

Round complexity

Captures the "execution time".

Complexity measures

Round complexity

Captures the “execution time”.

relevant parameter: D (diameter of G).

Complexity measures

Round complexity

Captures the "execution time".

relevant parameter: D (diameter of G).

Move complexity

Captures the "total workload".

Complexity measures

Round complexity

Captures the "execution time".

relevant parameter: D (diameter of G).

Move complexity

Captures the "total workload".

relevant parameter: n (the number of nodes of G).

Complexity measures

Round complexity

Captures the "execution time".

relevant parameter: D (diameter of G).

Move complexity

Captures the "total workload".

relevant parameter: n (the number of nodes of G).

Space complexity

Captures the local memory requirement.

Communication Model

Communication Model

Atomic State Model: Classical in self-stabilization [Dijkstra, 1974]

Communication Model

Atomic State Model: Classical in self-stabilization [Dijkstra, 1974]

Locally shared memory model with composite atomicity

Each node u has a **local state**.

Communication Model

Atomic State Model: Classical in self-stabilization [Dijkstra, 1974]

Locally shared memory model with composite atomicity

Each node u has a **local state**.

When moving, u **atomically**

- **reads** the states of its neighbors,
- **changes** its state.

Communication Model

Atomic State Model: Classical in self-stabilization [Dijkstra, 1974]

Locally shared memory model with composite atomicity

Each node u has a **local state**.

When moving, u **atomically**

- **reads** the states of its neighbors,
- **changes** its state.

Variants

- Nodes receive **sets**/multisets/... of states
- Nodes identified or **not identified**
- Ports labeled or **not labeled**

Litterature on Unison

	Rounds	Moves	Space	Daemon
Couvreur et al. (ICDCS'92)[2]	?	?	$\Theta(\log N)$	unfair

Finite memory implies the knowledge of $N \geq n$ or $B \geq D$.

[2] Not in Atomic State Model

Litterature on Unison

	Rounds	Moves	Space	Daemon
Couvreur et al. (ICDCS'92)[2]	?	?	$\Theta(\log N)$	unfair
Awerbuch et al. (STOC'93)[2]	$O(D)$	?	∞	unfair

Finite memory implies the knowledge of $N \geq n$ or $B \geq D$.

[2] Not in Atomic State Model

Litterature on Unison

	Rounds	Moves	Space	Daemon
Couvreur et al. (ICDCS'92)[2]	?	?	$\Theta(\log N)$	unfair
Awerbuch et al. (STOC'93)[2]	$O(D)$	?	∞	unfair
Boulinier et al. (PODC'04)	$O(n)$	$O(Dn^3)$	$\Theta(\log N)$	unfair

Finite memory implies the knowledge of $N \geq n$ or $B \geq D$.

[2] Not in Atomic State Model

Litterature on Unison

	Rounds	Moves	Space	Daemon
Couvreur et al. (ICDCS'92)[2]	?	?	$\Theta(\log N)$	unfair
Awerbuch et al. (STOC'93)[2]	$O(D)$	?	∞	unfair
Boulinier et al. (PODC'04)	$O(n)$	$O(Dn^3)$	$\Theta(\log N)$	unfair
Emek et Keren. (PODC'21)	$O(B^3)$	unbounded	$\Theta(\log B)$	fair

Finite memory implies the knowledge of $N \geq n$ or $B \geq D$.

[2]Not in Atomic State Model

Litterature on Unison

	Rounds	Moves	Space	Daemon
Couvreur et al. (ICDCS'92)[2]	?	?	$\Theta(\log N)$	unfair
Awerbuch et al. (STOC'93)[2]	$O(D)$	?	∞	unfair
Boulinier et al. (PODC'04)	$O(n)$	$O(Dn^3)$	$\Theta(\log N)$	unfair
Emek et Keren. (PODC'21)	$O(B^3)$	unbounded	$\Theta(\log B)$	fair
This paper	$\boldsymbol{O(D)}$	$\boldsymbol{O(n^3)}$	$\boldsymbol{\Theta(\log B)}$	**unfair**

Finite memory implies the knowledge of $N \geq n$ or $B \geq D$.
[2]Not in Atomic State Model

Consequences

	Rounds	Moves	Space
Unison	$2D+2$	$O(\min(n^2B, n^3))$	$\lceil \log B \rceil + 2$

With $B \geq 2D+2$

Consequences

	Rounds	Moves	Space
Unison	$2D+2$	$O(\min(n^2B, n^3))$	$\lceil \log B \rceil + 2$
Synchronizer	$5D+3T$	$O(\min(n^2B, n^3) + nT)$	$2M + \lceil \log B \rceil + 2$

Synchronizer input: self-stabilizing algorithm $\mathcal{A}$

T, M = synchronous time, space of $\mathcal{A}$.

With $B \geq 2D+2$

Consequences

	Rounds	Moves	Space
Unison	$2D+2$	$O(\min(n^2B, n^3))$	$\lceil \log B \rceil + 2$
Synchronizer	$5D+3T$	$O(\min(n^2B, n^3)+nT)$	$2M + \lceil \log B \rceil + 2$

Synchronizer input: self-stabilizing algorithm $\mathcal{A}$
T, M = synchronous time, space of $\mathcal{A}$.

Problem	Rounds	Moves	Space
BFS tree in rooted networks	$O(D)$	$O(n^3)$	$\Theta(\log B + \log \Delta)$
BFS tree in identified networks	$O(D)$	$O(n^3)$	$\Theta(\log N)$
Leader election	$O(D)$	$O(n^3)$	$\Theta(\log N)$
$O(\frac{n}{k})$-clustering	$O(D)$	$O(n^3)$	$\Theta(\log k + \log N)$

With $B \geq 2D+2$ and $N \geq n$

Questions?

State: $p.s \in \{C, E\}$, $p.c \in \begin{cases} [-B, B) & \text{if } p.s = C \\ [-B, 0) & \text{if } p.s = E \end{cases}$

Predicates:

$$\begin{aligned}
root(p) &:= \big(p.s = E \wedge \neg(\exists q \in N(p),\ q.s = E \wedge q.c < p.c)\big) \vee \\
&\quad \big(p.s = C \wedge \exists q \in N(p),\ (q.c \geq p.c + 2) \wedge \neg(p.c = 0 \wedge q.c = B - 1)\big) \\
activeRoot(p) &:= root(p) \wedge (p.c \neq -B \vee p.s = C) \\
errPropag(p, i) &:= i < 0 \wedge \exists q \in N(p),\ q.s = E \wedge q.c < i < p.c \\
canClearE(p) &:= p.s = E \wedge \forall q \in N(p),\ \big(|q.c - p.c| \leq 1 \wedge (q.c \leq p.c \vee q.s = C)\big) \\
updatable(p) &:= p.s = C \wedge \forall q \in N(p),\ q.c \in \{p.c, p.c \oplus_B 1\}
\end{aligned}$$

Rules:

$$\begin{array}{rcl|rcl}
R_R : activeRoot(p) & \rightarrow & (p.s, p.c) := (E, -B) & R_C : canClearE(p) & \rightarrow & p.s := C \\
R_P(i) : errPropag(p, i) & \rightarrow & (p.s, p.c) := (E, i) & R_U : updatable(p) & \rightarrow & p.c := p.c \oplus_B 1
\end{array}$$

R_R: highest priority, $R_P(i)$ higher priority that $R_P(i + l)$ for $l > 0$.