Concurrency in (Snap-Stabilizing) Local Resource Allocation

Karine Altisen, Stéphane Devismes, and Anaïs Durand

May 27, 2016
Resource Allocation

$n$ processes, $k$ resources, $n \gg k$
Resource Allocation

\[ n \text{ processes, } k \text{ resources, } n \gg k \]

Critical Section

Code executed when using resources. Lasts a finite, yet unbounded time.
Resource Allocation Problems

- Mutual Exclusion
  - **Safety:** At most one process in critical section at any time.
  - **Liveness:** Each requesting process enters its critical section in finite time.
Resource Allocation Problems

- **Mutual Exclusion**
  - **Safety**: At most one process in critical section at any time.
  - **Liveness**: Each requesting process enters its critical section in finite time.

- $\ell$-exclusion
- $k$-out-of-$\ell$-exclusion
- Group Mutual Exclusion
- Dining Philosophers
- Drinking Philosophers
- Reader-Writer
- ...
Resource Allocation Problems

- Mutual Exclusion
  - **Safety**: At most one process in critical section at any time.
  - **Liveness**: Each requesting process enters its critical section in finite time.

- $\ell$-exclusion
- $k$-out-of-$\ell$-exclusion
- Group Mutual Exclusion
- Dining Philosophers
- Drinking Philosophers
- Reader-Writer
- ...

Several resources ⇒

- **Concurrency**: Maximize the utilization rate of resources
  - Often not considered
\(\ell\)-exclusion [Fischer \textit{et al}, 79]

\(\ell\) identical copies of a non-shareable reusable resource.

- **Safety:** At most \(\ell\) processes can execute their critical section concurrently.
\(\ell\)-exclusion [Fischer et al, 79]

\(\ell\) identical copies of a non-shareable reusable resource.

- **Safety:** At most \(\ell\) processes can execute their critical section concurrently.
- **Liveness:** Each requesting process enters its critical section in finite time.
\( \ell \)-exclusion [Fischer et al, 79]

\( \ell \) identical copies of a non-shareable reusable resource.

- **Safety:** At most \( \ell \) processes can execute their critical section concurrently.

- **Liveness:** Each requesting process enters its critical section in finite time.

- **Avoiding \( \ell \)-deadlock:** If fewer than \( \ell \) processes are in their critical section, then it is possible for another (requesting) process to enter its critical section, even though no process leaves its critical section in the meantime.
Avoiding $\ell$-deadlock

$\ell = 4$
Avoiding $\ell$-deadlock

$\ell = 4$
Avoiding $\ell$-deadlock

$\ell = 4$
Avoiding $\ell$-deadlock

$\ell = 4$
Avoiding $\ell$-deadlock

$\ell = 4$
Avoiding $\ell$-Deadlock:
$\ell$-exclusion problem [Fischer et al, 79]

$(k, \ell)$-Liveness:
$k$-out-of-$\ell$-exclusion problem [Datta et al, 03]

Maximal-Concurrency:
Committee coordination problem [Bonakdarpour et al, 11]
Properties Handling Concurrency

- **Avoiding $\ell$-Deadlock:**
  $\ell$-exclusion problem [Fischer et al, 79]

- **$(k, \ell)$-Liveness:**
  $k$-out-of-$\ell$-exclusion problem [Datta et al, 03]

- **Maximal-Concurrency:**
  Committee coordination problem [Bonakdarpour et al, 11]

Drawback: dedicated to a specific problem
No Deadlock: $P_{Free} \neq \emptyset \Rightarrow$ algorithm not blocked even if no more request or exit from CS.

$P_{Free} = \{ \text{requesting processes that can be served without violating safety} \}$
Maximal-concurrency [Altisen et al, 15]

- **No Deadlock:** $P_{Free} \neq \emptyset \Rightarrow$ algorithm not blocked even if no more request or exit from CS.
- **No Livelock:**
  
  $P_{Free} \neq \emptyset$

  +

  no request and no exit from CS during sufficient long time

  $\Rightarrow$ someone served

$P_{Free} = \{ \text{requesting processes that can be served without violating safety} \}$
Alternative Definition of Maximal-concurrency [Altisen et al, 15]

- **No Deadlock**: $P_{Free} \neq \emptyset \Rightarrow$ algorithm not blocked even if no more request or exit from CS.

\[
P_{Free} = \{ \text{requesting processes that can be served without violating safety} \}
\]
Alternative Definition of Maximal-concurrency [Altisen et al, 15]

- **No Deadlock:** $P_{Free} \neq \emptyset \Rightarrow$ algorithm not blocked even if no more request or exit from CS.
- **No Livelock:**
  - no request and no exit from CS during sufficient long time $\Rightarrow P_{Free}$ becomes $\emptyset$

$P_{Free} = \{ \text{requesting processes that can be served without violating safety} \}$
Alternative Definition of Maximal-concurrency [Altisen et al, 15]

- **No Deadlock:** $P_{Free} \neq \emptyset \Rightarrow$ algorithm not blocked even if no more request or exit from CS.
- **No Livelock:**
  
  no request and no exit from CS during sufficient long time
  
  $\Rightarrow P_{Free}$ becomes $\emptyset$

\[P_{Free} = \{ \text{requesting processes that can be served without violating safety} \}\]

⚠️ **Not always possible:** [Datta et al, 03] in $k$-out-of-$\ell$ exclusion
Partial-concurrency w.r.t $\mathcal{P}$ [Altisen et al, 15]

- **No Deadlock**: algorithm not blocked except if $P_{Free} \subseteq X$
even if no more request or exit from CS

\[ P_{Free} = \{ \text{requesting processes that can be served without violating safety} \} \]
Partial-concurrency w.r.t $\mathcal{P}$ [Altisen et al, 15]

- **No Deadlock**: algorithm not blocked except if $P_{\text{Free}} \subseteq X$ even if no more request or exit from CS

  with $\mathcal{P}(X)$

- **No Livelock**: no request and no exit from CS during sufficient long time

  $\Rightarrow P_{\text{Free}}$ becomes $\subseteq X$

  with $\mathcal{P}(X)$

$$P_{\text{Free}} = \{ \text{requesting processes that can be served without violating safety} \}$$
Focus on Local Resource Allocation [Cantarell et al, 03]

Generalization of Many Classical Problems

- Dining Philosophers
- Local Mutual Exclusion
- Drinking Philosophers
- Local Reader-Writer
- Local Group Mutual Exclusion
- ...
Focus on Local Resource Allocation [Cantarell et al, 03]

- **$X \Leftrightarrow Y$:** Resource types $X$ and $Y$ are compatible.

- **Safety:** Two neighbors can concurrently execute their critical section using $X$ and $Y$, respectively, if $X \Leftrightarrow Y$.

- **Liveness:** Each requesting process enters its critical section in finite time.
Local Resource Allocation Instances

- **Dining Philosophers:**
  - One resource type:
  - ![Forks](image1)
  - ![Forks](image2)
Local Resource Allocation Instances

- **Dining Philosophers:**
  - One resource type:
  - ✗
  - ✗

- **Local Reader-Writer:**
  - Two resource types: 🔁, 🆇
  - 🔁 ✗
  - 🔁 ✗
  - 🆇 ✗
  - 🆇 ✗
Focus on Local Resource Allocation [Cantarell et al, 03]

- **$X \Leftrightarrow Y$:** Resource types $X$ and $Y$ are *compatible*.

- **Safety:** Two neighbors can concurrently execute their critical section using $X$ and $Y$, respectively, if $X \Leftrightarrow Y$.

- **Liveness:** Each requesting process enters its critical section in finite time.

⚠️ No property handling concurrency
Concurrent in Local Resource Allocation

Impossibility of Maximal-Concurrency

\[ q_0, q_i, q_j, q_1, q_2, \ldots \]

\[ x \not\equiv x \]

\[ x \not\equiv x \]
Concurrent imposibility of Maximal-Concurrency.

\[ \text{x \neq \text{ x}} \]
Concurrent in Local Resource Allocation

Impossibility of Maximal-Concurrency

\[
\begin{align*}
x \neq x 
\end{align*}
\]
Impossibility of Maximal-Concurrency

\[ x \neq x \]
Concurrent in Local Resource Allocation

Impossibility of Maximal-Concurrency

\[ \begin{align*}
    q_0 & \to p & q_1 & \to p & q_2 & \to p \\
    q_k & \to p & q_j & \to p & q_i & \to p
\end{align*} \]

\[ x \neq x \]
Concurrency in Local Resource Allocation

Impossibility of Maximal-Concurrency

\[ x \not\sim x \]

\[ p \] never served = liveness violated

⇒ Maximal-concurrency impossible

\[ \text{Strong-concurrency} = \text{Partial-concurrency} \, \text{w.r.t.} \, P_{\text{strong}} \]

\[ P_{\text{strong}}(X) \equiv X = (CN_p \setminus \{q\} \cup CN_q) \]

Anaïs Durand
Concurrency in Local Resource Allocation

Impossibility of Maximal-Concurrency

\[ x \neq x \]
Impossibility of Maximal-Concurrency

\[ x \not\Rightarrow x \]
Concurrent in Local Resource Allocation

Impossibility of Maximal-Concurrency

$p$ never served = liveness violated

$\Rightarrow$ Maximal-concurrency impossible
Concurrent in Local Resource Allocation

Strong-Concurrency

$p$ never served $= \text{liveness violated}$

$\Rightarrow \text{Maximal-concurrency impossible}$

- **Strong-concurrency** =
  Partial-concurrency \(w.r.t\ \mathcal{P}_{\text{strong}}\)

\[
\mathcal{P}_{\text{strong}}(X) \equiv X = (CN_p \setminus \{q\} \cup CN_q)
\]
Conclusion

- Definition of two properties of concurrency

- **Negative result:** Impossibility of maximal-concurrency in LRA

- **Positive result:** Design of a strong-concurrent LRA algorithm

**Perspectives:** Concurrency in other classes of resource allocation problems?
Thank you for your attention.

Do you have any questions?