Distributed Deadlock-Avoidance

César Sánchez

IMDEA Software Institute, Spain

DRV Workshop, Bertinoro
19-May, 2016
A little story about how static knowledge can help solve unsolvable problems

César Sánchez

IMDEA Software Institute, Spain

DRV Workshop, Bertinoro 19-May, 2016
Introduction

Goal: Formalization of middleware services
Deadlocks

Deadlock is one of the classical problems in CS

One (common) approach is the ostrich approach

The other approaches are: detection, prevention and avoidance.

<table>
<thead>
<tr>
<th></th>
<th>centralized</th>
<th>distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>detection</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>prevention</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>avoidance</td>
<td>Banker’s</td>
<td>impractical</td>
</tr>
</tbody>
</table>

Efficient dynamic resource allocation can have a big practical impact.
Distributed Dining Philosophers
Distributed Dining Philosophers

A deadlock state
Distributed Dining Philosophers

Detection:
Distributed Dining Philosophers

Prevention:
Distributed Dinning Philosophers

Avoidance:
Distributed Dinning Philosophers

Distributed Avoidance:
Deadlock Avoidance Problem Space

Centralized

Unsolvable

Distributed

Unsolvable
Deadlock Avoidance Problem Space

- Centralized
- Distributed

- Unsolvable
- Max utilization
- Unsolvable

- Unsolvable
Deadlock Avoidance Problem Space

Centralized

Unsolvable

Max utilization

[Dijkstra’65]

Distributed

Unsolvable

[Singhal’95]
Deadlock Avoidance Problem Space

Centralized

Unsolvable

Max utilization

[Dijkstra’65] [Singhal’95]

[de Alfaro+:05]

FMS

Distributed

Unsolvable
Deadlock Avoidance Problem Space

Centralized

Unsolvable

Max utilization

[Dijkstra’65] [Singhal’95]

[de Alfaro+:05]

FMS

Distributed

Unsolvable

?
Distributed Real-Time and Embedded

A B

C D

E F
Distributed Real-Time and Embedded

\[ n_1 \]
Distributed Real-Time and Embedded
Distributed Real-Time and Embedded
Distributed Real-Time and Embedded

Sequence of calls:

\[ n_1 \rightarrow A \rightarrow n_2 \rightarrow C \rightarrow n_3 \rightarrow B \rightarrow n_1 \]
\[ \quad \rightarrow n_2 \rightarrow C \rightarrow n_5 \rightarrow E \rightarrow n_1 \]
Distributed Real-Time Embedded Systems

Distributed Real-Time Embedded Systems:

- Asynchronous distributed system
- Limited Resources
- Wait-on-connection
- Arbitrary number of processes spawned
- All processes terminate

**Problem**: deadlocks are possible if no controller is used
Example of Deadlock

Two sites, with two resources each:

and the call graph:
Example of Deadlock

Two sites, with two resources each:

and the call graph:
Example of Deadlock

Two sites, with two resources each:

\[ \text{and the call graph:} \]

\[ \begin{align*}
  n_1 & \rightarrow A \\
  m_1 & \rightarrow B \\
  n_2 & \rightarrow B \\
  m_2 & \rightarrow A
\end{align*} \]
Example of Deadlock

Two sites, with two resources each:

and the call graph:

\[\begin{align*}
&n_1 \quad A \\
&m_1 \quad B \\
&n_2 \quad B \\
&m_2 \quad A
\end{align*}\]
Example of Deadlock

Two sites, with two resources each:

and the call graph:
Example of Deadlock

Two sites, with two resources each:

and the call graph:
Example of Deadlock

Two sites, with two resources each:

and the call graph:
## Summary of Contributions

<table>
<thead>
<tr>
<th>Contribution #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient deadlock Avoidance can is possible provided call-graphs are known statically</td>
</tr>
</tbody>
</table>
## Summary of Contributions

<table>
<thead>
<tr>
<th>Contribution #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient deadlock Avoidance can is possible provided call-graphs are known statically</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contribution #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal annotations can be efficiently computed. If annotations are not followed <em>anomalies</em> can occur.</td>
</tr>
</tbody>
</table>
## Summary of Contributions

<table>
<thead>
<tr>
<th>Contribution #1</th>
<th>Efficient deadlock Avoidance can is possible provided call-graphs are known statically</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contribution #2</td>
<td>Optimal annotations can be efficiently computed. If annotations are not followed <em>anomalies</em> can occur.</td>
</tr>
<tr>
<td>Contribution #3</td>
<td>Distributed Deadlock Avoidance with (individual) liveness guarantees can be efficiently achieved.</td>
</tr>
</tbody>
</table>
Model of Computation

- Remote procedure call (with Wait-On-Connection)
- Asynchronous messages
- All to all communication
- Finite resources: $T_A$ total number of threads
Model of Computation

- Remote procedure call (with Wait-On-Connection)
- Asynchronous messages
- All to all communication
- Finite resources: $T_A$ total number of threads

We seek a *deadlock avoidance* solution with no extra communication
Distributed Deadlock Avoidance Solution

Two parts:

1. Static:

2. Dynamic:
Distributed Deadlock Avoidance Solution

Two parts:

1. Static:

2. Dynamic:
Distributed Deadlock Avoidance Solution

Two parts:

1. Static:

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

2. Dynamic:

\[
\text{when } En \text{ do} \\
\begin{align*}
&En \\
&n_1() \\
&Out
\end{align*}
\]

\} entry section
\} method invocation
\} exit section
Annotations

Annotations are computed statically

\[
\begin{align*}
n_1 &\quad A &\quad 0 &\quad 0 \\
m_1 &\quad B &\quad 0 &\quad 0 \\
n_2 &\quad B &\quad 0 &\quad 0 \\
m_2 &\quad A &\quad 0 &\quad 0
\end{align*}
\]
Annotations

Annotations are computed statically

\[ n_1 \longrightarrow n_2 \]
\[ m_1 \longrightarrow m_2 \]

Dependency edges \( n \rightarrow m \) whenever \( \alpha(n) \leq m \) for two calls in the same node.
Annotations

Annotations are computed statically

\[ n \rightarrow m \]

whenever \( \alpha(n) \leq m \) for two calls in the same node.

\[ n_1 \quad A \quad n_2 \quad B \]

\[ m_1 \quad B \quad m_2 \quad A \]
Annotations

Annotations are computed statically

Dependency edges $n \to m$ whenever $\alpha(n) \leq m$ for two calls in the same node.

- $n$ depends on $m$ if there is a path from $n$ to $m$ containing a $\to$

- A dependency cycle is close path with a $\to$
Annotations

Annotations are computed statically

\[
\begin{align*}
\alpha(n) & \leq m \\
\end{align*}
\]

for two calls in the same node.

- \( n \) depends on \( m \) if there is a path from \( n \) to \( m \) containing a \( \rightarrow \)
- A dependency cycle is close path with a \( \rightarrow \)
Annotations

Annotations are computed statically

\[ n \rightarrow m \quad \text{whenever} \quad \alpha(n) \leq m \quad \text{for two calls in the same node.} \]

- \( n \) depends on \( m \) if there is a path from \( n \) to \( m \) containing a →
- A dependency cycle is close path with a →
Annotations

Annotations are computed statically

Dependency edges $n \rightarrow m$ whenever $\alpha(n) \leq m$ for two calls in the same node.

- $n$ depends on $m$ if there is a path from $n$ to $m$ containing a $\rightarrow$
- A dependency cycle is close path with a $\rightarrow$
Basic Solution Deadlock Avoidance

Protocol \texttt{Basic-P}: 

\[
\begin{bmatrix}
    \text{when } \alpha < t_A \text{ do} \\
    t_A-- \\
    n_1() \\
    t_A++
\end{bmatrix}
\]
Basic Solution Deadlock Avoidance

Protocol $\text{Basic-P}$: \( \alpha \)

\[
\begin{bmatrix}
 n_1 & A \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
 \text{when } \alpha < t_A \text{ do} \\
 t_A -- \\
 n_1() \\
 t_A ++ \\
\end{bmatrix}
\]

**Theorem:** If \( \alpha \) has no cyclic dependencies, then $\text{Basic-P}$ guarantees absence of deadlock.
Basic Solution Deadlock Avoidance

Protocol $\text{Basic-P}$: 

\[
\begin{array}{c}
\begin{array}{c}
\text{n}_1 \ A \\
\text{when } \alpha < t_A \text{ do} \\
\text{n}_1() \\
\text{t}_A++
\end{array}
\end{array}
\]

**Theorem:** If $\alpha$ has no cyclic dependencies, then $\text{Basic-P}$ guarantees absence of deadlock.

**Lemma:** The following is an invariant:

*The number of processes running methods with annotation $i$ or higher is at most $T_A - i$.*
The Annotation Theorem

**Theorem**: If $\alpha$ has no cyclic dependencies, then $\text{Basic-P}$ guarantees absence of deadlock.

**Lemma**: The following is an invariant:

*The number of processes running methods with annotation $i$ or higher is at most $T_A - i$.*

<table>
<thead>
<tr>
<th>annotation $\alpha$</th>
<th>Max num of procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$T_A$</td>
</tr>
<tr>
<td>1</td>
<td>$T_A - 1$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$T_A - 1$</td>
<td>1</td>
</tr>
</tbody>
</table>
The Annotation Theorem

**Theorem**: If $\alpha$ has no cyclic dependencies, then $\text{Basic-P}$ guarantees absence of deadlock.

**Lemma**: The following is an invariant:

*The number of processes running methods with annotation $i$ or higher is at most $T_A - i$.*

**Lemma**: If a request $\alpha$ is disabled, then there is an active process running $\alpha_2$ with $\alpha_2 \leq \alpha$. 
Annotations

Two immediate questions:

1. How to compute acyclic annotations

2. What if annotations are not acyclic?
Annotations

Two immediate questions:

1. How to compute acyclic annotations
   - Visit nodes following some reverse topological order.
   - When visiting $n$, compute the set of nodes $S$ previously visited and reachable following $(\rightarrow \cup \leftarrow)$*.
   - Set $\alpha(n)$ to 1 plus the largest node in $S$ that resides in the same site.

2. What if annotations are not acyclic?
Annotations

Two immediate questions:

1. How to compute acyclic annotations
   – Visit nodes following some reverse topological order.
   – When visiting \( n \), compute the set of nodes \( S \) previously visited and reachable following \((\rightarrow \cup \rightarrow)^*\).
   – Set \( \alpha(n) \) to 1 plus the largest node in \( S \) that resides in the same site.

2. What if annotations are not acyclic?

\[
\begin{align*}
T_A &= 1 \\
T_B &= 1 \\
T_C &= 1
\end{align*}
\]
Annotations

Two immediate questions:

1. How to compute acyclic annotations
   - Visit nodes following some reverse topological order.
   - When visiting $n$, compute the set of nodes $S$ previously visited and reachable following $(\rightarrow \cup \leftarrow \cdot)$*.
   - Set $\alpha(n)$ to 1 plus the largest node in $S$ that resides in the same site.

2. What if annotations are not acyclic?

FACT: Given enough resources, a deadlock is reachable

\[
\begin{align*}
T_A &= 1 \\
T_B &= 1 \\
T_C &= 1
\end{align*}
\]
How about liveness?

Consider two nodes, with two resource each \((T_A = T_B = 2)\):

\[ t_A = 2 \quad \text{and} \quad t_B = 2 \]

and the call graph:

```
       0
       n_1 A ----> n_2 B
       1
       m_1 B ----> m_2 A
```
How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

\[ t_A = 2 \quad \text{and} \quad t_B = 2 \]

and the call graph:
How about liveness?

Consider two nodes, with two resource each \((T_A = T_B = 2)\):

\[
\begin{align*}
A & \quad t_A = 0 \\
B & \quad t_B = 2
\end{align*}
\]

and the call graph:

\[
\begin{align*}
m_1 \quad B & \quad n_1 \quad A \\
m_2 \quad A & \quad n_2 \quad B
\end{align*}
\]
How about liveness?

Consider two nodes, with two resource each \((T_A = T_B = 2)\):

\[
\begin{align*}
\text{Node } A & : t_A = 0 \\
\text{Node } B & : t_B = 2
\end{align*}
\]

and the call graph:

\[
\begin{align*}
\text{Call graph}
\end{align*}
\]
How about liveness?

Consider two nodes, with two resource each \((T_A = T_B = 2)\):

\[ t_A = 0 \quad \text{and} \quad t_B = 0 \]

and the call graph:

\[ n_1 \xrightarrow{0} A \xrightarrow{0} n_2 \quad B \xrightarrow{0} m_1 \xrightarrow{1} B \xrightarrow{0} m_2 \xrightarrow{0} A \]
How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

$A$ and $B$

$t_A = 0$  
$t_B = 1$

and the call graph:

$\begin{array}{c}
\begin{array}{c}
\text{A} \\
\text{m_1} \quad \text{B}
\end{array}
\rightarrow
\begin{array}{c}
\text{n_2} \quad \text{B}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\text{n_1} \quad \text{A}
\end{array}
\rightarrow
\begin{array}{c}
\text{m_2} \quad \text{A}
\end{array}
\end{array}$
How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

and the call graph:
How about liveness?

Consider two nodes, with two resource each \((T_A = T_B = 2)\):

- Node A: \(t_A = 0\)
- Node B: \(t_B = 1\)

And the call graph:

```
  n1   A   n2    B
    0       0

  m1   B   m2    A
    ×   1       0
```
How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

A \quad B

$t_A = 0 \quad t_B = 0$

and the call graph:

\[
\begin{array}{c}
\text{n}_1 \quad \text{A} \\
\text{m}_1 \quad \text{B} \\
\end{array}
\quad \begin{array}{c}
\text{n}_2 \quad \text{B} \\
\text{m}_2 \quad \text{A} \\
\end{array}
\]
Revisiting the Invariant

**Lemma:** The following is an invariant:

The number of processes running methods with annotation $i$ or higher is at most $T_A - i$.

$$\text{act}_{A, \geq i} \leq T_A - i \quad \text{for all notes } A \text{ and } i$$

where

- $\text{act}_{A, i}$ : number of active processes in $A$ with annotation $i$
- $\text{act}_{A, \geq i} = \sum_{k \geq i} \text{act}_{A, i}$
Revisiting the Invariant

**Lemma:** The following is an invariant:

*The number of processes running methods with annotation $i$ or higher is at most $T_A - i$.*

$$act_{A,\geq i} \leq T_A - i \quad \text{for all notes } A \text{ and } i$$

where

- $act_{A,i}$ : number of active processes in $A$ with annotation $i$
- $act_{A,\geq i} = \sum_{k \geq i} act_{A,i}$

The weakest precondition on allowing a request for $\square_{n A} i$

$$\varphi' = \bigwedge_k \left\{ \begin{array}{ll}
act_{A,\geq k} & \leq T_A - k \quad \text{if } k > i \\
act_{A,\geq k} + 1 & \leq T_A - k \quad \text{if } k \leq i
\end{array} \right.$$
**The Protocol Live-P**

To execute $n_A^i$:

\[
\begin{bmatrix}
\text{when } \varphi' \text{ do} \\
act_{A,i}++ \\
n() \\
act_{A,i}--
\end{bmatrix}
\]

**Theorem** (Deadlock Avoidance): If $\alpha$ is acyclic, then Live-P guarantees absence of deadlock.

**Theorem** (Liveness): If $\alpha$ is acyclic, then Live-P guarantees that every waiting process is eventually enabled.
Live-P provides liveness

Consider two nodes, with two resource each ($T_A = T_B = 2$):

\[
\begin{align*}
A & : t_A = 2 \\
B & : t_B = 2
\end{align*}
\]

\[
\begin{align*}
\text{LIVE-P} & : \\
& \begin{array}{c}
& n_1 A \\ m_1 B
& \end{array} \rightarrow \begin{array}{c}
& n_2 B \\ m_2 A
& \end{array} \\
& \begin{array}{c}
act_B, \geq 0 \\
act_B, \geq 1
& \end{array}
\end{align*}
\]

\[
\begin{align*}
\text{BASIC-P} & : \\
& \begin{array}{c}
& n_1 A \\ m_1 B
& \end{array} \rightarrow \begin{array}{c}
& n_2 B \\ m_2 A
& \end{array} \\
& \begin{array}{c}
act_B, \geq 0 \\
act_B, \geq 1
& \end{array}
\end{align*}
\]
Live-P provides liveness

Consider two nodes, with two resource each \((T_A = T_B = 2)\):

\[
t_A = 2 \\
B \\
B
\]

**Live-P**

\[
\begin{array}{ccc}
n_1 & A & \rightarrow & n_2 & B \\
m_1 & B & \rightarrow & m_2 & A \\
act_B, \geq 0 \\
\end{array}
\]

**Basic-P**

\[
\begin{array}{ccc}
n_1 & A & \rightarrow & n_2 & B \\
m_1 & B & \rightarrow & m_2 & A \\
act_B, \geq 1 \\
\end{array}
\]
Live-P provides liveness

Consider two nodes, with two resource each ($T_A = T_B = 2$):

$$t_A = 0 \quad t_B = 2$$

**Live-P**

- $n_1 A \rightarrow n_2 B$
- $m_1 B \rightarrow m_2 A$
- $act_B, \geq 0$

**Basic-P**

- $n_1 A \rightarrow n_2 B$
- $m_1 B \rightarrow m_2 A$
- $act_B, \geq 1$
Live-P provides liveness

Consider two nodes, with two resource each ($T_A = T_B = 2$):

\[ t_A = 0 \quad t_B = 2 \]

**Live-P**

\[
\begin{array}{c}
\text{n}_1 \quad \text{A} \\
\text{m}_1 \quad \text{B} \\
\text{act}_B, \geq 0
\end{array}
\rightarrow
\begin{array}{c}
\text{n}_2 \quad \text{B} \\
\text{m}_2 \quad \text{A} \\
\text{act}_B, \geq 1
\end{array}
\]

**Basic-P**

\[
\begin{array}{c}
\text{n}_1 \quad \text{A} \\
\text{m}_1 \quad \text{B} \\
\text{act}_B, \geq 0
\end{array}
\rightarrow
\begin{array}{c}
\text{n}_2 \quad \text{B} \\
\text{m}_2 \quad \text{A} \\
\text{act}_B, \geq 1
\end{array}
\]
Live-P provides liveness

Consider two nodes, with two resource each ($T_A = T_B = 2$):

\[
\begin{align*}
   t_A &= 0 \\
   t_B &= 0
\end{align*}
\]

\[
\begin{align*}
   n_1 A & \rightarrow n_2 B \\
   m_1 B & \rightarrow m_2 A
\end{align*}
\]

\[
\begin{align*}
   \text{act}_B, \geq 0 \\
   \text{act}_B, \geq 1
\end{align*}
\]
Live-P provides liveness

Consider two nodes, with two resources each ($T_A = T_B = 2$):

$$t_A = 0$$  \hspace{1cm}  $$t_B = 1$$

**Live-P**

- $n_1 \xrightarrow{0} n_2$
- $m_1 \xrightarrow{1} m_2$

- $act_{B, \geq 0}$

**Basic-P**

- $n_1 \xrightarrow{0} n_2$
- $m_1 \xrightarrow{1} m_2$

- $act_{B, \geq 1}$
Conclusions

- Distributed Deadlock Avoidance is possible without communication
- ... provided call-graphs are known
- Using static annotations + runtime protocols
- If cycles are allowed (e.g. by uncontrolled resource allocation), then deadlocks are unavoidable, provided enough resources
- Individual liveness is also enforceable

Future work:
- is deadlock avoidance enforceable for any amount of initial resources?
- can this be adapted to composable conveyor systems?
Conclusions

- Distributed Deadlock Avoidance is possible without communication
- ... provided call-graphs are known
- Using static annotations + runtime protocols
- If cycles are allowed (e.g. by uncontrolled resource allocation), then deadlocks are unavoidable
- Individual liveness is also enforceable

Future work:
- is deadlock avoidance enforceable for any amount of initial resources?
- can this be adapted to composable conveyor systems?
Distributed Dining Philosophers

Distributed Avoidance:
Distributed Dinning Philosphers

Distributed Avoidance:

SOLUTION:
Distributed Dinning Philosphers

Distributed Avoidance:

SOLUTION:
For your first pick,
do not the take last fork if going in increasing order.

For your second pick,
do as you wish.
Thank you for your attention!