Strategies for Replica Placement in Tree Networks

Anne Benoit, Veronika Rehn and Yves Robert

GRAAL team, LIP
École Normale Supérieure de Lyon

October 2006
Replica placement in tree networks

Set of clients (tree leaves): requests with QoS constraints, known in advance

Internal nodes may be provided with a replica; in this case they become servers and process requests (up to their capacity limit)

How many replicas required?
Which locations?
Total replica cost?
Replica placement in tree networks

Set of clients (tree leaves): requests with QoS constraints, known in advance

Internal nodes may be provided with a replica; in this case they become servers and process requests (up to their capacity limit)

How many replicas required?
Which locations?
Total replica cost?
Replica placement in tree networks

Set of clients (tree leaves): requests with QoS constraints, known in advance

Internal nodes may be provided with a replica; in this case they become servers and process requests (up to their capacity limit)

How many replicas required?
Which locations?
Total replica cost?
Introduction and motivation

- Replica placement in tree networks
- Set of clients (tree leaves): requests with QoS constraints, known in advance
- Internal nodes may be provided with a replica; in this case they become servers and process requests (up to their capacity limit)

How many replicas required?
Which locations?
Total replica cost?
Rule of the game

- Handle all client requests, and minimize cost of replicas
- \( \rightarrow \text{REPLICA PLACEMENT problem} \)
- Several policies to assign replicas

![Diagram of a tree with nodes labeled 1, 2, 2, 3, 5, 4, 3.}]
Rule of the game

- Handle all client requests, and minimize cost of replicas
- \( \rightarrow \text{REPLICA PLACEMENT problem} \)
- Several policies to assign replicas

![Diagram](image)
Rule of the game

- Handle all client requests, and minimize cost of replicas
- $\Rightarrow$ REPLICA PLACEMENT problem
- Several policies to assign replicas

\[
W = 10
\]
Rule of the game

- Handle all client requests, and minimize cost of replicas
- \( \rightarrow \text{Replica Placement problem} \)
- Several policies to assign replicas

\[ W = 10 \]
Rule of the game

- Handle all client requests, and minimize cost of replicas
- \( \rightarrow \) REPLICA PLACEMENT problem
- Several policies to assign replicas
Rule of the game

- Handle all client requests, and minimize cost of replicas
- \( \rightarrow \text{REPLICA PLACEMENT problem} \)
- Several policies to assign replicas
Major contributions

**Theory**  
New access policies  
Problem complexity  
LP-based lower bound to cost of REPLICA PLACEMENT

**Practice**  
Heuristics for each policy  
Experiments to assess impact of new policies
Major contributions

**Theory**
- New access policies
- Problem complexity
- LP-based lower bound to cost of **Replica Placement**

**Practice**
- Heuristics for each policy
- Experiments to assess impact of new policies
Outline

1. Framework
2. Access policies
3. Complexity results
4. Linear programming formulation
5. Heuristics for the Replica Cost problem
6. Experiments
7. Extensions
8. Conclusion
Outline

1. Framework
2. Access policies
3. Complexity results
4. Linear programming formulation
5. Heuristics for the Replica Cost problem
6. Experiments
7. Extensions
8. Conclusion
Definitions and notations

- Distribution tree $T$, clients $C$ (leaf nodes), internal nodes $N$

- **Client** $i \in C$:
  - Sends $r_i$ requests per time unit (number of accesses to a single object database)
  - Quality of service $q_i$ (response time)

- **Node** $j \in N$:
  - Can contain the object database replica (server) or not
  - Processing capacity $W_j$
  - Storage cost $sc_j$

- **Tree edge**: $l \in \mathcal{L}$ (communication link between nodes)
  - Communication time $\text{comm}_l$
  - Bandwidth limit $BW_l$
Definitions and notations

- **Distribution tree** $\mathcal{T}$, clients $\mathcal{C}$ (leaf nodes), internal nodes $\mathcal{N}$

- **Client** $i \in \mathcal{C}$:
  - Sends $r_i$ requests per time unit (number of accesses to a single object database)
  - Quality of service $q_i$ (response time)

- **Node** $j \in \mathcal{N}$:
  - Can contain the object database replica (server) or not
  - Processing capacity $W_j$
  - Storage cost $s_{cj}$

- **Tree edge**: $l \in \mathcal{L}$ (communication link between nodes)
  - Communication time $\text{comm}_l$
  - Bandwidth limit $\text{BW}_l$
Definitions and notations

- Distribution tree \( T \), clients \( C \) (leaf nodes), internal nodes \( N \)
- **Client** \( i \in C \):
  - Sends \( r_i \) requests per time unit (number of accesses to a single object database)
  - Quality of service \( q_i \) (response time)
- **Node** \( j \in N \):
  - Can contain the object database replica (server) or not
  - Processing capacity \( W_j \)
  - Storage cost \( sc_j \)
- **Tree edge**: \( l \in L \) (communication link between nodes)
  - Communication time \( \text{comm}_l \)
  - Bandwidth limit \( \text{BW}_l \)
Definitions and notations

- Distribution tree $\mathcal{T}$, clients $\mathcal{C}$ (leaf nodes), internal nodes $\mathcal{N}$

- **Client** $i \in \mathcal{C}$:
  - Sends $r_i$ requests per time unit (number of accesses to a single object database)
  - Quality of service $q_i$ (response time)

- **Node** $j \in \mathcal{N}$:
  - Can contain the object database replica (server) or not
  - Processing capacity $W_j$
  - Storage cost $sc_j$

- **Tree edge**: $l \in \mathcal{L}$ (communication link between nodes)
  - Communication time $comm_l$
  - Bandwidth limit $BW_l$
Tree notations

- $r$: tree root
- $\text{children}(j)$: set of children of node $j \in \mathcal{N}$
- $\text{parent}(k)$: parent in the tree of node $k \in \mathcal{N} \cup \mathcal{C}$
- link $l : k \rightarrow \text{parent}(k) = k'$. Then $\text{succ}(l)$ is the link $k' \rightarrow \text{parent}(k')$ (when it exists)
- Ancestors($k$): set of ancestors of node $k$
- If $k' \in \text{Ancestors}(k)$, then $\text{path}[k \rightarrow k']$: set of links in the path from $k$ to $k'$
- $\text{subtree}(k)$: subtree rooted in $k$, including $k$. 

Yves.Robert@ens-lyon.fr October 2006
Problem instances

- **Goal:** place replicas to process client requests
- **Client** $i \in \mathcal{C}$: $\text{Servers}(i) \subseteq \mathcal{N}$ set of servers responsible for processing its requests
- $r_{i,s}$: number of requests from client $i$ processed by server $s$ ($\sum_{s \in \text{Servers}(i)} r_{i,s} = r_i$)
- $R = \{ s \in \mathcal{N} | \exists i \in \mathcal{C}, s \in \text{Servers}(i) \}$: set of replicas
Constraints

- **Server capacity** – $\forall s \in R, \sum_{i \in \mathcal{C}, s \in \text{Servers}(i)} r_{i,s} \leq W_s$

- **QoS** – $\forall i \in \mathcal{C}, \forall s \in \text{Servers}(i), \sum_{l \in \text{path}[i \rightarrow s]} \text{comm}_l \leq q_i$.

- **Link capacity** – $\forall l \in \mathcal{L}, \sum_{i \in \mathcal{C}, s \in \text{Servers}(i)} r_{i,s} \leq \text{BW}_l$
Objective function

- $\text{Min } \sum_{s \in R} s c_s$
- Restrict to case where $s c_s = W_s$
- **Replica Cost** problem: no QoS nor bandwidth constraints; heterogeneous servers
- **Replica Counting** problem: idem, but homogeneous platforms
Objective function

- Min $\sum_{s \in R} sc_s$
- Restrict to case where $sc_s = W_s$
- **Replica Cost** problem: no QoS nor bandwidth constraints; heterogeneous servers
- **Replica Counting** problem: idem, but homogeneous platforms
Outline

1. Framework
2. Access policies
3. Complexity results
4. Linear programming formulation
5. Heuristics for the Replica Cost problem
6. Experiments
7. Extensions
8. Conclusion
Single server vs Multiple servers

**Single server** – Each client $i$ is assigned a single server $server(i)$, that is responsible for processing all its requests.

**Multiple servers** – A client $i$ may be assigned several servers in a set $Servers(i)$. Each server $s \in Servers(i)$ will handle a fraction $r_{i,s}$ of the requests.

In the literature: single server policy with additional constraint.
**Closest policy**

- **Closest**: single server policy

  Server of client $i$ is constrained to be first server found on the path that goes from $i$ upwards to the tree root

  Consider a client $i$ and its server $\text{server}(i)$:
  $$\forall i' \in \text{subtree}(\text{server}(i)), \quad \text{server}(i') \in \text{subtree}(\text{server}(i))$$

  Requests from $i'$ cannot “traverse” $\text{server}(i)$ and be served higher
New policies not studied in the literature

*Upwards*: Closest constraint is relaxed

*Multiple*: relax single server restriction

Expect more solutions with new policies, at a lower cost

*QoS constraints* may lower difference between policies
Example: existence of a solution

(a): solution for all policies
(b): no solution with Closest
(c): no solution with Closest nor Upwards
Example: existence of a solution

(a): solution for all policies
(b): no solution with Closest
(c): no solution with Closest nor Upwards
Example: existence of a solution

- (a): solution for all policies
- (b): no solution with Closest
- (c): no solution with Closest nor Upwards
Example: existence of a solution

- (a): solution for all policies
- (b): no solution with $Closest$
- (c): no solution with $Closest$ nor $Upwards$

$$W = 1$$
Upwards versus Closest

- **Upwards**: 3 replicas in $s_{2n}$, $s_{2n+1}$ and $s_{2n+2}$
- **Closest**: at least $n + 2$ replicas (replica in $s_{2n+1}$ or not)

$W = n$
**Replica Counting:** *Multiple* twice better than *Upwards*.

- Performance ratio: open problem.

Multiple: $n + 1$ replicas / Upwards: $2n$ replicas
**Replica Counting:** Multiple twice better than Upwards.

Performance ratio: open problem.

Multiple: \( n + 1 \) replicas / Upwards: \( 2n \) replicas
**Replica Cost:** *Multiple* arbitrarily better than *Upwards*

\[ s_1, W_1 = n \]

\[ s_2, W_2 = n \]

\[ n - 1 \]

\[ s_3, W_3 = Kn \]

\[ n + 1 \]

*Multiple:* cost $2n$ / *Upwards:* cost $(K + 1)n$
Lower bound for the **Replica Counting** problem

Obvious lower bound: \[\left\lceil \frac{\sum_{i \in C} r_i}{W} \right\rceil = 2\]

All policies require \(n + 1\) replica (one at each node).
Lower bound for the \textbf{Replica Counting} problem

Obvious lower bound: \[
\left\lceil \frac{\sum_{i \in C} r_i}{W} \right\rceil = 2
\]

All policies require \( n + 1 \) replica (one at each node).
Outline

1. Framework
2. Access policies
3. Complexity results
4. Linear programming formulation
5. Heuristics for the Replica Cost problem
6. Experiments
7. Extensions
8. Conclusion
### Complexity results - Basic problem

<table>
<thead>
<tr>
<th>Replica Counting</th>
<th>Replica Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Homogeneous</strong></td>
<td></td>
</tr>
<tr>
<td>Closest</td>
<td>Polynomial [Cidon02, Liu06]</td>
</tr>
<tr>
<td>Upwards</td>
<td></td>
</tr>
<tr>
<td>Multiple</td>
<td></td>
</tr>
<tr>
<td><strong>Heterogeneous</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Complexity results for the different instances of the problem

- *Closest/Homogeneous:* only known result (Cidon et al. 2002, Liu et al. 2006)
### Complexity results - Basic problem

<table>
<thead>
<tr>
<th></th>
<th>Replica Counting</th>
<th>Replica Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Homogeneous</td>
<td>Heterogeneous</td>
</tr>
<tr>
<td>Closest Upwards</td>
<td>polynomial [Cidon02, Liu06]</td>
<td></td>
</tr>
<tr>
<td>Multiple</td>
<td>polynomial algorithm</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Complexity results for the different instances of the problem

- **Closest/Homogeneous:** only known result (Cidon et al. 2002, Liu et al. 2006)
- **Multiple/Homogeneous:** nice algorithm to prove polynomial complexity
Complexity results - Basic problem

<table>
<thead>
<tr>
<th>Replica Counting</th>
<th>Replica Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Homogeneous</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Closest</strong></td>
<td>polynomial [Cidon02,Liu06]</td>
</tr>
<tr>
<td><strong>Upwards</strong></td>
<td>NP-complete</td>
</tr>
<tr>
<td><strong>Multiple</strong></td>
<td>polynomial algorithm</td>
</tr>
<tr>
<td><strong>Heterogeneous</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table: Complexity results for the different instances of the problem

- **Closest**/Homogeneous: only known result (Cidon et al. 2002, Liu et al. 2006)
- **Multiple**/Homogeneous: nice algorithm to prove polynomial complexity
- **Upwards**/Homogeneous: surprisingly, NP-complete
Complexity results - Basic problem

<table>
<thead>
<tr>
<th></th>
<th>Replica Counting</th>
<th>Replica Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Homogeneous</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Closest</td>
<td>polynomial [Cidon02,Liu06]</td>
<td>NP-complete</td>
</tr>
<tr>
<td>Upwards</td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>Multiple</td>
<td>polynomial algorithm</td>
<td>NP-complete</td>
</tr>
<tr>
<td><strong>Heterogeneous</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NP-complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NP-complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

Table: Complexity results for the different instances of the problem

- *Closest*/Homogeneous: only known result (Cidon et al. 2002, Liu et al. 2006)
- *Multiple*/Homogeneous: nice algorithm to prove polynomial complexity
- *Upwards*/Homogeneous: surprisingly, NP-complete
- All instances for the Heterogeneous case are NP-complete
Complexity results - QoS and Bandwidth

- **Closest/Homogeneous + QoS**: Polynomial (Liu et al.)
- **Closest/Homogeneous + Bandwidth**: ?? (Probably polynomial, Vero works at it 😊)
- **Multiple/Homogeneous + QoS**: NP-complete (reduction to 2-partition)
- **Multiple/Homogeneous + Bandwidth**: Polynomial? Algorithm quite similar to the case without BW, but proof still to check.
Complexity results - QoS and Bandwidth

- **Closest/Homogeneous + QoS**: Polynomial (Liu et al.)
- **Closest/Homogeneous + Bandwidth**: ?? (Probably polynomial, Vero works at it 😊)
- **Multiple/Homogeneous + QoS**: NP-complete (reduction to 2-partition)
- **Multiple/Homogeneous + Bandwidth**: Polynomial? Algorithm quite similar to the case without BW, but proof still to check.
Complexity results - QoS and Bandwidth

- Closest/Homogeneous + QoS: Polynomial (Liu et al.)
- Closest/Homogeneous + Bandwidth: ?? (Probably polynomial, Vero works at it 😊)
- Multiple/Homogeneous + QoS: NP-complete (reduction to 2-partition)
- Multiple/Homogeneous + Bandwidth: Polynomial? Algorithm quite similar to the case without BW, but proof still to check.
Complexity results - QoS and Bandwidth

- **Closest/Homogeneous + QoS**: *Polynomial* (Liu et al.)
- **Closest/Homogeneous + Bandwidth**: ?? (Probably *polynomial*, Vero works at it 😊)
- **Multiple/Homogeneous + QoS**: *NP-complete* (reduction to 2-partition)
- **Multiple/Homogeneous + Bandwidth**: *Polynomial?* Algorithm quite similar to the case without BW, but proof still to check.
Multiple/Homogeneous: greedy algorithm

3-pass algorithm:
- Select nodes which can handle \( W \) requests
- Select some extra servers to fulfill remaining requests
- Decide which requests are processed where

Example to illustrate algorithm (informally)

Proof of optimality: any optimal solution can be transformed into a solution similar to the one of the algorithm (moving requests from one server to another)
Multiple/Homogeneous: greedy algorithm

3-pass algorithm:
- Select nodes which can handle \( W \) requests
- Select some extra servers to fulfill remaining requests
- Decide which requests are processed where

Example to illustrate algorithm (informally)

Proof of optimality: any optimal solution can be transformed into a solution similar to the one of the algorithm (moving requests from one server to another)
Multiple/Homogeneous: greedy algorithm

3-pass algorithm:

- Select nodes which can handle W requests
- Select some extra servers to fulfill remaining requests
- Decide which requests are processed where

Example to illustrate algorithm (informally)

Proof of optimality: any optimal solution can be transformed into a solution similar to the one of the algorithm (moving requests from one server to another)
Multiple/Homogeneous: example

Initial network

The example network
Multiple/Homogeneous: example

Pass 1

Placing saturated replicas
Multiple/Homogeneous: example

Pass 2

Placing extra replicas: $n_4$ has maximum useful flow
Multiple/Homogeneous: example

Pass 2
Placing extra replicas: $n_2$ is of maximum useful flow 1
Multiple/Homogeneous: example

Pass 3

Deciding where requests are processed
The Replica Counting problem with the Upwards strategy is NP-complete in the strong sense.

Reduction from 3-PARTITION

\[ \sum_{i=1}^{3m} a_i = mB \]
The Replica Counting problem with the Upwards strategy is NP-complete in the strong sense

Reduction from 3-PARTITION

\[ \sum^{3m}_{i=1} a_i = mB \]
Heterogeneous network: **REPLICA COST** problem

- All three instances of the **REPLICA COST** problem with heterogeneous nodes are NP-complete
- Reduction from 2-PARTITION

\[\sum_{i=1}^{m} a_i = S, \quad a_{m+1} = 1, \quad W_j = a_i, \quad W_r = S/2 + 1\]

Solution with total storage cost \(S + 1\)?
Heterogeneous network: **Replica Cost problem**

- All three instances of the **Replica Cost** problem with heterogeneous nodes are NP-complete
- Reduction from 2-PARTITION

Mathematical expression:

\[
\sum_{i=1}^{m} a_i = S, \quad a_{m+1} = 1, \quad W_j = a_i, \quad W_r = S/2 + 1
\]

Solution with total storage cost \( S + 1 \)?
All three instances of the Replica Cost problem with heterogeneous nodes are NP-complete

Reduction from 2-PARTITION

\[ \sum_{i=1}^{m} a_i = S, \quad a_{m+1} = 1, \quad W_j = a_i, \quad W_r = S/2 + 1 \]

Solution with total storage cost \( S + 1 \)?
Outline

1. Framework
2. Access policies
3. Complexity results
4. Linear programming formulation
5. Heuristics for the Replica Cost problem
6. Experiments
7. Extensions
8. Conclusion
Linear programming

- **General instance** of the problem
  - Heterogeneous tree
  - QoS and bandwidth constraints
  - *Closest, Upwards* and *Multiple* policies

- **Integer linear program**: no efficient algorithm

- **Absolute lower bound** if program solved over the rationals (using the GLPK software)

- *Closest/Upwards* LP formulation
Linear programming

- **General instance** of the problem
  - Heterogeneous tree
  - QoS and bandwidth constraints
  - *Closest, Upwards* and *Multiple* policies

- **Integer linear program**: no efficient algorithm

- **Absolute lower bound** if program solved over the rationals
  (using the **GLPK** software)

- **Closest/Upwards** LP formulation
**Linear program: variables**

- $x_j$: boolean variable equal to 1 if $j$ is a server (for one or several clients)
- $y_{i,j}$: boolean variable equal to 1 if $j = \text{server}(i)$
  - If $j \notin \text{Ancests}(i)$, $y_{i,j} = 0$
- $z_{i,l}$: boolean variable equal to 1 if link $l \in \text{path}[i \rightarrow r]$ used when $i$ accesses server($i$)
  - If $l \notin \text{path}[i \rightarrow r]$, $z_{i,l} = 0$

**Objective function:** \[ \sum_{j \in N} \text{sc}_j x_j \]
Linear program: variables

- $x_j$: boolean variable equal to 1 if $j$ is a server (for one or several clients)
- $y_{i,j}$: boolean variable equal to 1 if $j = \text{server}(i)$
  - If $j \notin \text{Ancests}(i)$, $y_{i,j} = 0$
- $z_{i,l}$: boolean variable equal to 1 if link $l \in \text{path}[i \rightarrow r]$ used when $i$ accesses server($i$)
  - If $l \notin \text{path}[i \rightarrow r]$, $z_{i,l} = 0$

Objective function: $\sum_{j \in \mathcal{N}} \text{sc}_j x_j$
Linear program: variables

- **$x_j$:** boolean variable equal to 1 if $j$ is a server (for one or several clients)
- **$y_{i,j}$:** boolean variable equal to 1 if $j = \text{server}(i)$
  - If $j \notin \text{Ancests}(i)$, $y_{i,j} = 0$
- **$z_{i,l}$:** boolean variable equal to 1 if link $l \in \text{path}[i \rightarrow r]$ used when $i$ accesses server($i$)
  - If $l \notin \text{path}[i \rightarrow r]$, $z_{i,l} = 0$

Objective function: $\sum_{j \in N} s_c j \cdot x_j$
Linear program: variables

- $x_j$: boolean variable equal to 1 if $j$ is a server (for one or several clients)
- $y_{i,j}$: boolean variable equal to 1 if $j = \text{server}(i)$
  - If $j \notin \text{Ancests}(i)$, $y_{i,j} = 0$
- $z_{i,l}$: boolean variable equal to 1 if link $l \in \text{path}[i \rightarrow r]$ used when $i$ accesses server($i$)
  - If $l \notin \text{path}[i \rightarrow r]$, $z_{i,l} = 0$

Objective function: $\sum_{j \in N} sc_j x_j$
Linear program: constraints

- **Servers**: \( \forall i \in \mathcal{C}, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1 \)
- **Links**: \( \forall i \in \mathcal{C}, z_{i,i \to \text{parent}(i)} = 1 \)
- **Conservation**: \( \forall i \in \mathcal{C}, \forall l : j \to j' = \text{parent}(j) \in \text{path}[i \to r], z_{i,\text{succ}(l)} = z_{i,l} - y_{i,j'} \)
- **Server capacity**: \( \forall j \in \mathcal{N}, \sum_{i \in \mathcal{C}} r_i y_{i,j} \leq W_j x_j \)
- **Bandwidth limit**: \( \forall l \in \mathcal{L}, \sum_{i \in \mathcal{C}} r_i z_{i,l} \leq BW_l \)
- **QoS constraint**: \( \forall i \in \mathcal{C}, \forall j \in \text{Ancestors}(i), \text{dist}(i,j)y_{i,j} \leq q_i \)
- **Closest constraint**: \( \forall i \in \mathcal{C}, \forall j \in \text{Ancestors}(i) \setminus \{r\}, \forall i' \in \mathcal{C} \cap \text{subtree}(j), y_{i,j} + z_{i',j \to \text{parent}(j)} \leq 1 \)
Linear program: constraints

- **Servers**: $\forall i \in C, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1$
- **Links**: $\forall i \in C, z_{i,\text{parent}(i)} = 1$
- **Conservation**: $\forall i \in C, \forall l : j \rightarrow j' = \text{parent}(j) \in \text{path}[i \rightarrow r], z_{i,\text{succ}(l)} = z_{i,l} - y_{i,j'}$
- **Server capacity**: $\forall j \in N, \sum_{i \in C} r_i y_{i,j} \leq W_j x_j$
- **Bandwidth limit**: $\forall l \in L, \sum_{i \in C} r_i z_{i,l} \leq BW_l$
- **QoS constraint**: $\forall i \in C, \forall j \in \text{Ancestors}(i), \text{dist}(i,j) y_{i,j} \leq q_i$
- **Closest constraint**: $\forall i \in C, \forall j \in \text{Ancestors}(i) \setminus \{r\}$, $\forall i' \in C \cap \text{subtree}(j), y_{i,j} + z_{i',\text{parent}(j)} \leq 1$
Linear program: constraints

- **Servers**: $\forall i \in C, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1$
- **Links**: $\forall i \in C, z_{i,i \rightarrow \text{parent}(i)} = 1$
- **Conservation**: $\forall i \in C, \forall l : j \rightarrow j' = \text{parent}(j) \in \text{path}[i \rightarrow r], \quad z_{i,\text{succ}(l)} = z_{i,l} - y_{i,j'}$
- Server capacity: $\forall j \in \mathcal{N}, \sum_{i \in C} r_i y_{i,j} \leq W_j x_j$
- Bandwidth limit: $\forall l \in \mathcal{L}, \sum_{i \in C} r_i z_{i,l} \leq \text{BW}_l$
- QoS constraint: $\forall i \in C, \forall j \in \text{Ancestors}(i), \text{dist}(i,j) y_{i,j} \leq q_i$
- Closest constraint: $\forall i \in C, \forall j \in \text{Ancestors}(i) \setminus \{r\}, \forall i' \in C \cap \text{subtree}(j), y_{i,j} + z_{i',j \rightarrow \text{parent}(j)} \leq 1$
Linear program: constraints

- **Servers**: \( \forall i \in C, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1 \)
- **Links**: \( \forall i \in C, z_{i,i \rightarrow \text{parent}(i)} = 1 \)
- **Conservation**: \( \forall i \in C, \forall l: j \rightarrow j' = \text{parent}(j) \in \text{path}[i \rightarrow r], z_{i,\text{succ}(l)} = z_{i,l} - y_{i,j'} \)
- **Server capacity**: \( \forall j \in N, \sum_{i \in C} r_i y_{i,j} \leq W_j x_j \)
- **Bandwidth limit**: \( \forall l \in L, \sum_{i \in C} r_i z_{i,l} \leq BW_l \)
- **QoS constraint**: \( \forall i \in C, \forall j \in \text{Ancestors}(i), \text{dist}(i,j) y_{i,j} \leq q_i \)
- **Closest constraint**: \( \forall i \in C, \forall j \in \text{Ancestors}(i) \setminus \{r\}, \forall i' \in C \cap \text{subtree}(j), y_{i,j} + z_{i',j \rightarrow \text{parent}(j)} \leq 1 \)
Linear program: constraints

- **Servers**: $\forall i \in C, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1$
- **Links**: $\forall i \in C, z_{i,i \rightarrow \text{parent}(i)} = 1$
- **Conservation**: $\forall i \in C, \forall l : j \rightarrow j' = \text{parent}(j) \in \text{path}[i \rightarrow r], z_{i,\text{succ}(l)} = z_{i,l} - y_{i,j'}$
- **Server capacity**: $\forall j \in N, \sum_{i \in C} r_i y_{i,j} \leq W_j x_j$
- **Bandwidth limit**: $\forall l \in L, \sum_{i \in C} r_i z_{i,l} \leq BW_l$
- **QoS constraint**: $\forall i \in C, \forall j \in \text{Ancestors}(i), \text{dist}(i,j) y_{i,j} \leq q_i$
- **Closest constraint**: $\forall i \in C, \forall j \in \text{Ancestors}(i) \setminus \{r\}, \forall i' \in C \cap \text{subtree}(j), y_{i,j} + z_{i',j \rightarrow \text{parent}(j)} \leq 1$
Linear program: constraints

- **Servers:** $\forall i \in C, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1$
- **Links:** $\forall i \in C, z_{i,i \rightarrow \text{parent}(i)} = 1$
- **Conservation:** $\forall i \in C, \forall l : j \rightarrow j' = \text{parent}(j) \in \text{path}[i \rightarrow r], z_{i,\text{succ}(l)} = z_{i,l} - y_{i,j'}$
- **Server capacity:** $\forall j \in N, \sum_{i \in C} r_i y_{i,j} \leq W_j x_j$
- **Bandwidth limit:** $\forall l \in L, \sum_{i \in C} r_i z_{i,l} \leq BW_l$
- **QoS constraint:** $\forall i \in C, \forall j \in \text{Ancestors}(i), \text{dist}(i,j) y_{i,j} \leq q_i$
- **Closest constraint:** $\forall i \in C, \forall j \in \text{Ancestors}(i) \setminus \{r\}, \forall i' \in C \cap \text{subtree}(j), y_{i,j} + z_{i',j \rightarrow \text{parent}(j)} \leq 1$
Linear program: constraints

- **Servers**: \( \forall i \in C, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1 \)
- **Links**: \( \forall i \in C, z_{i,i \rightarrow \text{parent}(i)} = 1 \)
- **Conservation**: \( \forall i \in C, \forall l : j \rightarrow j' = \text{parent}(j) \in \text{path}[i \rightarrow r],
  
  z_{i,\text{succ}(l)} = z_{i,l} - y_{i,j'} \)
- **Server capacity**: \( \forall j \in N, \sum_{i \in C} r_i y_{i,j} \leq W_j x_j \)
- **Bandwidth limit**: \( \forall l \in L, \sum_{i \in C} r_i z_{i,l} \leq BW_l \)
- **QoS constraint**: \( \forall i \in C, \forall j \in \text{Ancestors}(i), \text{dist}(i,j) y_{i,j} \leq q_i \)
- **Closest constraint**: \( \forall i \in C, \forall j \in \text{Ancestors}(i) \setminus \{r\},
  \forall i' \in C \cap \text{subtree}(j), y_{i,j} + z_{i',j \rightarrow \text{parent}(j)} \leq 1 \)
Multiple formulation

- Similar formulation, with
  - \( y_{i,j} \): integer variable = nb requests from client \( i \) processed by node \( j \)
  - \( z_{i,l} \): integer variable = nb requests flowing through link \( l \)
- Constraints are slightly modified
An ILP-based lower bound

- **Solving over the rationals**: solution for all practical values of the problem size
  - Not very precise bound
  - *Upwards/Closest* equivalent to *Multiple* when solved over the rationals
- Integer solving: limitation to $s \leq 50$ nodes and clients
- Mixed bound obtained by solving the *Upwards* formulation over the rational and imposing only the $x_j$ being integers
  - Resolution for problem sizes $s \leq 400$
  - Improved bound: if a server is used only at 50% of its capacity, the cost of placing a replica at this node is not halved as it would be with $x_j = 0.5$.  

Yves.Robert@ens-lyon.fr October 2006
An ILP-based lower bound

- **Solving over the rationals**: solution for all practical values of the problem size
  - Not very precise bound
  - *Upwards/Closest* equivalent to *Multiple* when solved over the rationals
- **Integer solving**: limitation to $s \leq 50$ nodes and clients
  - Mixed bound obtained by solving the *Upwards* formulation over the rational and imposing only the $x_j$ being integers
    - Resolution for problem sizes $s \leq 400$
    - Improved bound: if a server is used only at 50% of its capacity, the cost of placing a replica at this node is not halved as it would be with $x_j = 0.5$. 
An ILP-based lower bound

- **Solving over the rationals**: solution for all practical values of the problem size
  - Not very precise bound
  - Upwards/Closest equivalent to Multiple when solved over the rationals

- **Integer solving**: limitation to $s \leq 50$ nodes and clients
- **Mixed bound** obtained by solving the Upwards formulation over the rational and imposing only the $x_j$ being integers
  - Resolution for problem sizes $s \leq 400$
  - **Improved bound**: if a server is used only at 50% of its capacity, the cost of placing a replica at this node is not halved as it would be with $x_j = 0.5$.  

Yves.Robert@ens-lyon.fr October 2006
Outline

1. Framework
2. Access policies
3. Complexity results
4. Linear programming formulation
5. **Heuristics for the Replica Cost problem**
6. Experiments
7. Extensions
8. Conclusion
Polynomial heuristics for the Replica Cost problem

- Heterogeneous platforms
- No QoS nor bandwidth constraints

- Experimental assessment of the relative performance of the three policies
- Traversals of the tree, bottom-up or top-down
- Worst case complexity $O(s^2)$, where $s = |\mathcal{C}| + |\mathcal{N}|$ is problem size
Heuristics

- Polynomial heuristics for the **Replica Cost** problem
  - Heterogeneous platforms
  - No QoS nor bandwidth constraints

- Experimental assessment of the relative performance of the three policies

- Traversals of the tree, bottom-up or top-down

- Worst case complexity $O(s^2)$,
  where $s = |C| + |N|$ is problem size
Polynomial heuristics for the **Replica Cost** problem

- Heterogeneous platforms
- No QoS nor bandwidth constraints

Experimental assessment of the relative performance of the three policies

- Traversals of the tree, bottom-up or top-down
- Worst case complexity $O(s^2)$, where $s = |C| + |N|$ is problem size
Heuristics for Closest

- Closest Top Down All **CTDA**
  - Breadth-first traversal of the tree
  - When a node can process the requests of all the clients in its subtree, node chosen as a server and exploration of the subtree stopped
  - Procedure called until no more servers are added
  - Choosing $n_2$, $n_4$ and then $n_1$
Heuristics for Closest

- Closest Top Down All CTDA
  - Breadth-first traversal of the tree
  - When a node can process the requests of all the clients in its subtree, node chosen as a server and exploration of the subtree stopped
  - Procedure called until no more servers are added
  - Choosing $n_2$, $n_4$ and then $n_1$
Heuristics for *Closest*

- Closest Top Down All **CTDA**
- Closest Top Down Largest First **CTDLF**
Heuristics for Closest

- Closest Top Down All **CTDA**
- Closest Top Down Largest First **CTDLF**
  - Traversal of the tree, treating subtrees that contains most requests first
  - When a node can process the requests of all the clients in its subtree, node chosen as a server and traversal stopped
  - Procedure called until no more servers are added
  - Choosing \( n_2 \) and then \( n_1 \)
Heuristics for *Closest*

- Closest Top Down All **CTDA**
- Closest Top Down Largest First **CTDLF**
- Closest Bottom Up **CBU**
Heuristics for Closest

- Closest Top Down All CTDA
- Closest Top Down Largest First CTDLF
- Closest Bottom Up CBU
  - Bottom-up traversal of the tree
  - When a node can process the requests of all the clients in its subtree, node chosen as a server
  - Choosing $n_3, n_5, n_1$
Heuristics for **Upwards**

- **Upwards Top Down** **UTD**
  - 2-pass algorithm
  - Select first saturating nodes, then extra nodes
  - Choosing $n_2$ (for $c_1$) and in second pass $n_1$ (for $c_2, c_3$)

![Diagram of UTD algorithm with nodes and edges]
Upwards Top Down **UTD**

- 2-pass algorithm
- Select first saturating nodes, then extra nodes
- Choosing $n_2$ (for $c_1$) and in second pass $n_1$ (for $c_2, c_3$)
Heuristics for *Upwards*

- Upwards Top Down **UTD**
- Upwards Big Client First **UBCF**
Heuristics for *Upwards*

- Upwards Top Down **UTD**
- Upwards Big Client First **UBCF**
  - Sorting clients by decreasing request numbers, and finding the server of minimal available capacity to process its requests.
  - Choosing $n_2$ for $c_1$, $n_1$ for $c_2$ and $n_1$ for $c_3$
Heuristics for *Multiple*

- A top-down and a bottom-up heuristic in 2-passes (MTD, MBU)

- A greedy heuristic MG, similar to Pass 3 of the polynomial algorithm for *Multiple/Homogeneous*: fill all servers as much as possible in a bottom-up fashion
Heuristics for *Multiple*

- A top-down and a bottom-up heuristic in 2-passes (MTD, MBU)
- A greedy heuristic MG, similar to Pass 3 of the polynomial algorithm for *Multiple*/Homogeneous: fill all servers as much as possible in a bottom-up fashion
A top-down and a bottom-up heuristic in 2-passes (MTD, MBU)

A greedy heuristic MG, similar to Pass 3 of the polynomial algorithm for Multiple/Homogeneous: fill all servers as much as possible in a bottom-up fashion

MG affects 4 requests to $n_2$, and then the remaining 2 requests to $n_1$
Heuristics for *Multiple*

- A top-down and a bottom-up heuristic in 2-passes (*MTD, MBU*)

- A greedy heuristic *MG*, similar to Pass 3 of the polynomial algorithm for *Multiple*/Homogeneous: fill all servers as much as possible in a bottom-up fashion

![Diagram of a tree with nodes and edges](attachment:image.png)

- MG affects 4 requests to $n_2$, and then the remaining 2 requests to $n_1$

- **CTDLF better on this example**: selects $n_1$ only
A top-down and a bottom-up heuristic in 2-passes (MTD, MBU)

A greedy heuristic **MG**, similar to Pass 3 of the polynomial algorithm for *Multiple*/Homogeneous: fill all servers as much as possible in a bottom-up fashion

**Heuristic MixedBest MB** which picks up best result over all heuristics: solution for the *Multiple* policy
Outline

1. Framework
2. Access policies
3. Complexity results
4. Linear programming formulation
5. Heuristics for the Replica Cost problem
6. Experiments
7. Extensions
8. Conclusion
Plan of experiments

- Assess impact of the different access policies
- Assess performance of the polynomial heuristics

Important parameter:

\[ \lambda = \frac{\sum_{i \in C} r_i}{\sum_{j \in N} W_i} \]

- 30 trees for each \( \lambda = 0.1, 0.2, \ldots, 0.9 \)
- Problem size \( s = |C| + |N| \) such that \( 15 \leq s \leq 400 \)
- Computation of the LP lower bound for each tree
Plan of experiments

- Assess impact of the different **access policies**
- Assess performance of the **polynomial heuristics**
- Important parameter:

\[
\lambda = \frac{\sum_{i \in C} r_i}{\sum_{j \in N} W_i}
\]

- **30 trees** for each \( \lambda = 0.1, 0.2, \ldots, 0.9 \)
- **Problem size** \( s = |C| + |N| \) such that \( 15 \leq s \leq 400 \)
- Computation of the **LP lower bound** for each tree
Results - Percentage of success

- **Number of solutions** for each lambda and each heuristic
- No LP solution → No solution for any heuristic
- Homogeneous case

![Graph showing the percentage of success for different heuristics and lambda values.](image-url)
Heterogeneous trees: similar results

- Striking impact of new policies
- MG and MB always find the solution
Results - Percentage of success

- Heterogeneous trees: similar results

- Striking impact of new policies
  - MG and MB always find the solution
Heterogeneous trees: similar results

Striking impact of new policies
MG and MB always find the solution
Results - Solution cost

- Distance of the result (in terms of replica cost) of the heuristic to the lower bound
- $T_\lambda$: subset of trees with a solution
- Relative cost:
  \[
  rcost = \frac{1}{|T_\lambda|} \sum_{t \in T_\lambda} \frac{cost_{LP}(t)}{cost_h(t)}
  \]

- $cost_{LP}(t)$: lower bound cost on tree $t$
- $cost_h(t)$: heuristic cost on tree $t$; $cost_h(t) = +\infty$ if $h$ did not find any solution
Results - Solution cost

- Distance of the result (in terms of replica cost) of the heuristic to the lower bound
- \( T_\lambda \): subset of trees with a solution
- Relative cost:

\[
rcost = \frac{1}{|T_\lambda|} \sum_{t \in T_\lambda} \frac{\text{cost}_{LP}(t)}{\text{cost}_h(t)}
\]

- \( \text{cost}_{LP}(t) \): lower bound cost on tree \( t \)
- \( \text{cost}_h(t) \): heuristic cost on tree \( t \); \( \text{cost}_h(t) = +\infty \) if \( h \) did not find any solution
Homogeneous results

![Chart showing various heuristics and their relative cost across different lambda values. Each heuristic is represented by a different line and marker. The heuristics include ClosestTopDownAll, ClosestTopDownLargestFirst, ClosestBottomUp, UpwardsTopDown, UpwardsBigClientFirst, MultipleGreedy, MultipleTopDown, MultipleBottomUp, and MixedBest. The y-axis represents the relative cost, while the x-axis represents the lambda value.](chart.png)
Heterogeneous results - similar to the homogeneous case

Results - Solution cost

- ClosestTopDownAll
- ClosestTopDownLargestFirst
- ClosestBottomUp
- UpwardsTopDown
- UpwardsBigClientFirst
- MultipleGreedy
- MultipleTopDown
- MultipleBottomUp
- MixedBest
Striking effect of new policies: many more solutions to the Replica Placement problem

Multiple $\geq$ Upwards $\geq$ Closest: hierarchy observed within our heuristics

Best Multiple heuristic (MB) always at 85% of the lower bound: satisfactory result
Outline

1. Framework
2. Access policies
3. Complexity results
4. Linear programming formulation
5. Heuristics for the **Replica Cost** problem
6. Experiments

7. Extensions
8. Conclusion
Extensions

- Simplified problem instance for this work
- Possible generalizations:
  - Several objects
  - More complex objective function
Extensions - Several objects

- We considered a single object: all replicas are identical
- Different types of objects need to be accessed: clients have requests of different types

- New parameters:
  - Requests per object $r_i^k$, and $q_i^{(k)}$
  - Size of the object, computation time involved, storage cost, ...

- Constraints and objective function slightly modified
- Constraints/Objective function add up linearly for different objects: LP-formulation easily extended.

- Efficient heuristics in this case: challenging problem
Extensions - Several objects

- We considered a single object: all replicas are identical.
- Different types of objects need to be accessed: clients have requests of different types.
- New parameters:
  - Requests per object $r_i^k$, and $q_i^{(k)}$.
  - Size of the object, computation time involved, storage cost, ...
- Constraints and objective function slightly modified.
- Constraints/Objective function add up linearly for different objects: LP-formulation easily extended.
- Efficient heuristics in this case: challenging problem.
We considered a single object: all replicas are identical

Different types of objects need to be accessed: clients have requests of different types

New parameters:
- Requests per object $r^k_i$, and $q^{(k)}_i$
- Size of the object, computation time involved, storage cost, ...

Constraints and objective function slightly modified

Constraints/Objective function add up linearly for different objects: LP-formulation easily extended.

Efficient heuristics in this case: challenging problem
Extensions - Several objects

- We considered a single object: all replicas are identical
- Different types of objects need to be accessed: clients have requests of different types

  - New parameters:
    - Requests per object $r_i^k$, and $q_i^{(k)}$
    - Size of the object, computation time involved, storage cost, ...

- Constraints and objective function slightly modified
- Constraints/Objective function add up linearly for different objects: LP-formulation easily extended.

- Efficient heuristics in this case: challenging problem
Extensions - Objective function

Cost of replica – What we considered in this work

Communication cost – This cost is the read cost

Update cost – The write cost is the extra cost due to an update of the replicas

Linear combination – A quite general objective function can be obtained by a linear combination of the three different costs

\[ \alpha \sum \text{replica cost} + \beta \sum \text{read cost} + \gamma \sum \text{write cost} \]

Efficient heuristics in this case: challenging problem
Extensions - Objective function

**Cost of replica** – What we considered in this work

**Communication cost** – This cost is the *read* cost

**Update cost** – The *write* cost is the extra cost due to an update of the replicas

**Linear combination** – A quite general objective function can be obtained by a linear combination of the three different costs

\[
\alpha \sum_{\text{servers, objects}} \text{replica cost} + \beta \sum_{\text{requests}} \text{read cost} + \gamma \sum_{\text{updates}} \text{write cost}
\]

Efficient heuristics in this case: challenging problem
Outline

1. Framework
2. Access policies
3. Complexity results
4. Linear programming formulation
5. Heuristics for the Replica Cost problem
6. Experiments
7. Extensions
8. Conclusion
Related work

- Several papers on replica placement, but...
  - ...all consider only the Closest policy

- Replica Placement in a general graph is NP-complete

- Wolfson and Milo: impact of the write cost, use of a minimum spanning tree for updates. Tree networks: polynomial solution


- Kalpakis et al: NP-completeness of a variant with bidirectional links (requests served by any node in the tree)

- Karlsson et al: comparison of different objective functions and several heuristics. No QoS, but several other constraints.

- Tang et al: real QoS constraints

- Rodolakis et al: Multiple policy but in a very different context
Several papers on replica placement, but...

...all consider only the *Closest* policy

Replica Placement in a general graph is NP-complete

Wolfson and Milo: impact of the *write* cost, use of a minimum spanning tree for updates. Tree networks: polynomial solution

Cidon et al (multiple objects) and Liu et al (QoS constraints): polynomial algorithms for homogeneous networks.

Kalpakis et al: NP-completeness of a variant with bidirectional links (requests served by any node in the tree)

Karlsson et al: comparison of different objective functions and several heuristics. No QoS, but several other constraints.

Tang et al: real QoS constraints

Rodolakis et al: *Multiple* policy but in a very different context
Related work

- Several papers on replica placement, but...
- ...all consider only the *Closest* policy

**Replica Placement** in a general graph is NP-complete

- Wolfson and Milo: impact of the *write* cost, use of a minimum spanning tree for updates. Tree networks: polynomial solution
- Kalpakis et al: NP-completeness of a variant with bidirectional links (requests served by any node in the tree)
- Karlsson et al: comparison of different objective functions and several heuristics. No QoS, but several other constraints.
- Tang et al: real QoS constraints
- Rodolakis et al: *Multiple* policy but in a very different context
Conclusion

- Introduction of two new policies for the Replica Placement problem
  - *Upwards* and *Multiple*: natural variants of the standard *Closest* approach → surprising they have not already been considered

Theoretical side — Complexity of each policy, for homogeneous and heterogeneous platforms

Practical side
  - Design of several heuristics for each policy
  - Comparison of their performance
  - Striking impact of the policy on the result
  - Use of a LP-based lower bound to assess the absolute performance, which turns out to be quite good.

Yves.Robert@ens-lyon.fr October 2006
Conclusion

- Introduction of two new policies for the **Replica Placement** problem

- *Upwards* and *Multiple*: natural variants of the standard *Closest* approach → surprising they have not already been considered

**Theoretical side** – Complexity of each policy, for homogeneous and heterogeneous platforms

**Practical side**

- Design of several heuristics for each policy
- Comparison of their performance
- Striking impact of the policy on the result
- Use of a LP-based lower bound to assess the absolute performance, which turns out to be quite good.
Future work

Short term

- More simulations for the Replica Cost problem: shape of the trees, distribution law of the requests, degree of heterogeneity of the platforms
- Designing heuristics for more general instances of the Replica Placement problem (QoS and bandwidth constraints): these constraints may lower the difference between policies

Longer term

- Consider the problem with several object types
- Extension with more complex objective functions

Still a lot of challenging algorithmic problems 😊
Future work

Short term

- More simulations for the Replica Cost problem: shape of the trees, distribution law of the requests, degree of heterogeneity of the platforms
- Designing heuristics for more general instances of the Replica Placement problem (QoS and bandwidth constraints): these constraints may lower the difference between policies

Longer term

- Consider the problem with several object types
- Extension with more complex objective functions

Still a lot of challenging algorithmic problems 😊