Strategies for Replica Placement
in Tree Networks

Anne Benoit, Veronika Rehn and Yves Robert

GRAAL team, LIP
École Normale Supérieure de Lyon

November 9, 2006

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?

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?

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?

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)
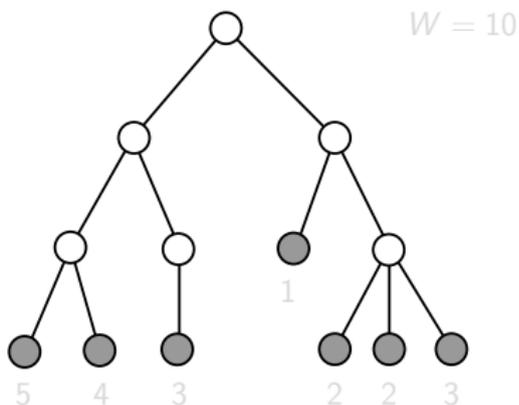
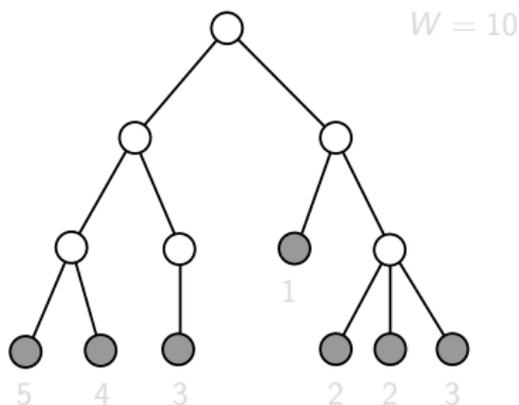How many replicas required?
Which locations?
Total replica cost?

Rule of the game

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

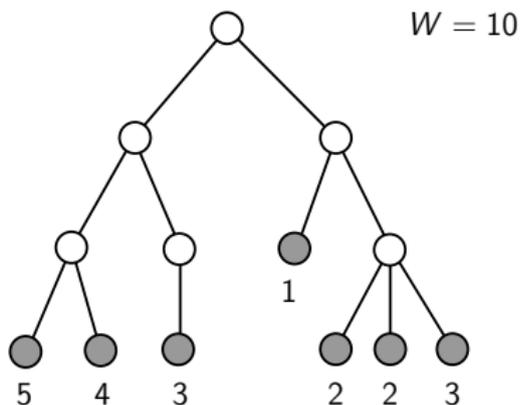$W = 10$

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
- $\rightarrow$ REPLICA PLACEMENT problem
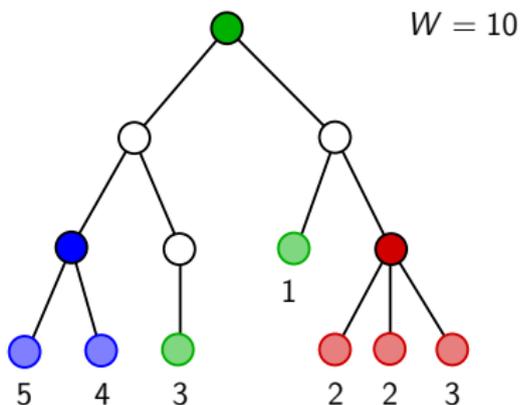- Several policies to assign replicas

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
- $\rightarrow$ REPLICA PLACEMENT problem
- Several policies to assign replicas
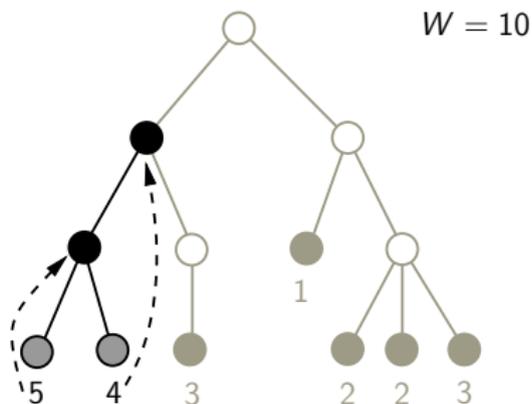

$W = 10$

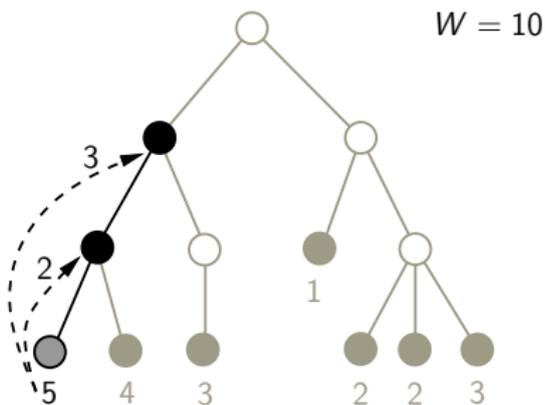Rule of the game

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

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

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 $\mathcal{T}$, 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

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

Intro **Framework** Policies Complexity LP Heuristics Experiments Extensions Conclusion
0000   ●0000   00000000 000000   00000 0000 0000000   000   000

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

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

Problem instances

- Goal: place replicas to process client requests
- Client $i \in \mathcal{C}$: 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 ($\sum_{s \in \text{Servers}(i)} r_{i,s} = r_i$)
- $R = \{s \in \mathcal{N} \mid \exists i \in C , s \in \text{Servers}(i)\}$: set of replicas

Intro 0000 **Framework** 00000 Policies 00000000 Complexity 000000 LP 00000 Heuristics 0000 Experiments 0000000 Extensions 000 Conclusion 000

Constraints

- **Server capacity** – $\forall s \in R, \sum_{i \in \mathcal{C} | s \in \mathsf{Servers}(i)} r_{i,s} \leq \mathsf{W}_s$

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

- **Link capacity** – $\forall l \in \mathcal{L} \sum_{i \in \mathcal{C}, s \in \mathsf{Servers}(i) | l \in \mathsf{path}[i \rightarrow s]} r_{i,s} \leq \mathsf{BW}_l$

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

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.
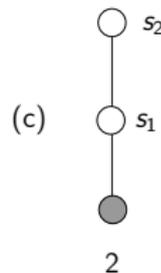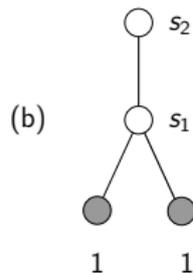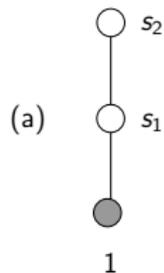
Closest policy

- *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 server(i):
  $\forall i' \in$ subtree(server(i)),   server(i') $\in$ subtree(server(i))
- Requests from i' cannot "traverse" server(i) and be served higher
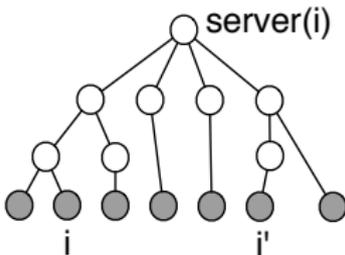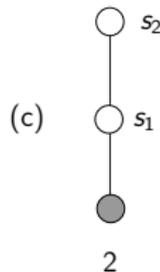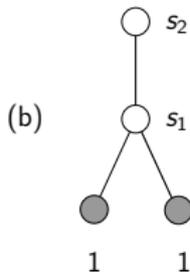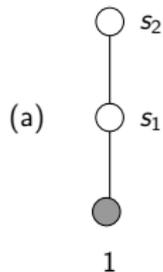
Upwards and *Multiple* policy

- 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

Intro   Framework   **Policies**   Complexity   LP   Heuristics   Experiments   Extensions   Conclusion
0000   00000   ○○○●○○○○   ○○○○○○   00000 0000   ○○○○○○○   ○○○   ○○○

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

$W = 1$

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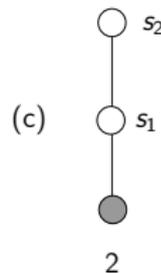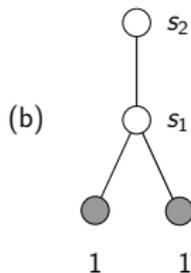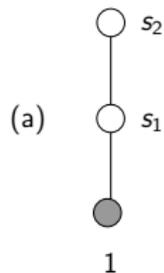
Example: existence of a solution

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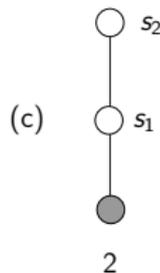- (c): no solution with *Closest* nor *Upwards*

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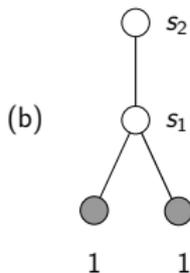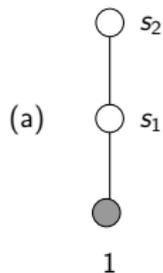
- (a): solution for all policies
- (b): no solution with *Closest*
- (c): no solution with *Closest* nor *Upwards*

Upwards versus Closest
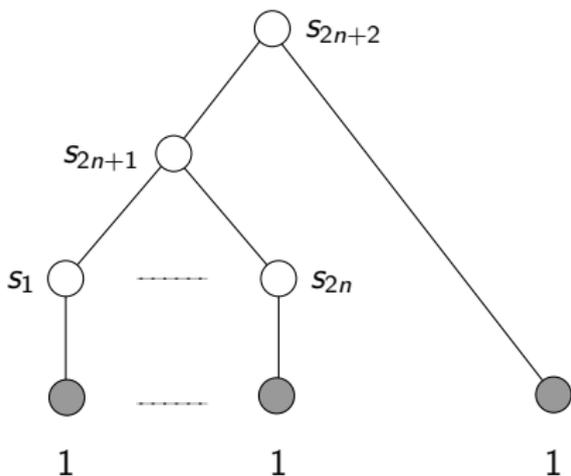
$W = n$

- *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)

Multiple versus Upwards

- REPLICA COUNTING: *Multiple* twice better than *Upwards*.
- Performance ratio: open problem.
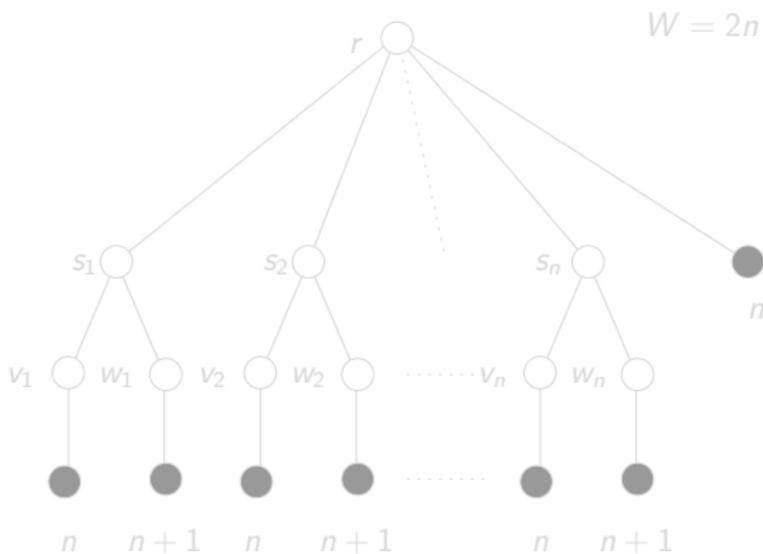
Multiple: $n + 1$ replicas / *Upwards*: $2n$ replicas

Multiple versus Upwards

- REPLICA COUNTING: *Multiple* twice better than *Upwards*.
- Performance ratio: open problem.
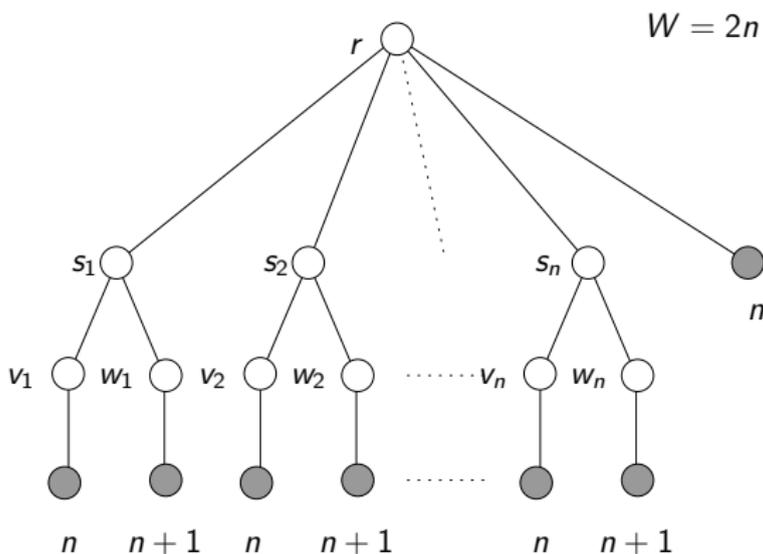
Multiple: $n + 1$ replicas / *Upwards*: $2n$ replicas

Multiple versus Upwards

- REPLICA COST: *Multiple* arbitrarily better than *Upwards*
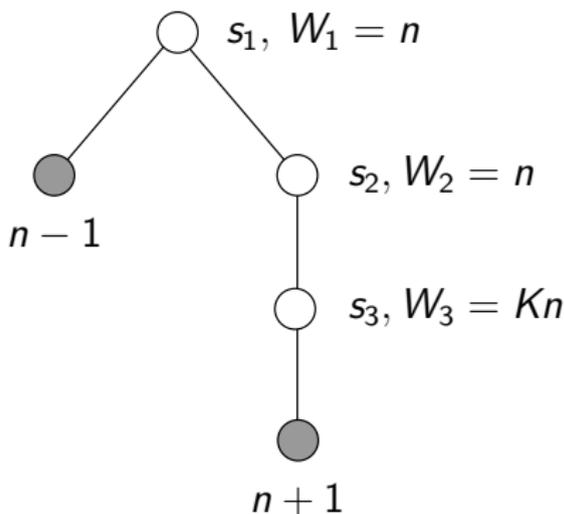
Multiple: cost $2n$ / *Upwards*: cost $(K+1)n$

Lower bound for the REPLICA COUNTING problem

Obvious lower bound: $\left\lceil \frac{\sum_{i \in \mathcal{C}} r_i}{W} \right\rceil = 2$
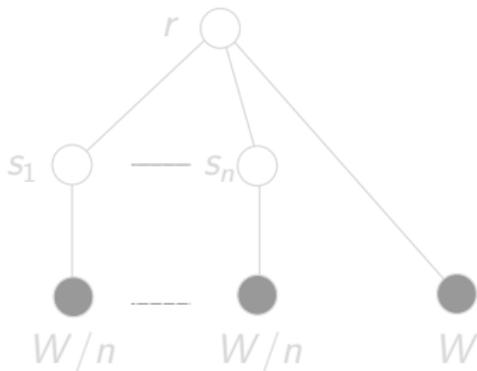
All policies require $n + 1$ replica (one at each node).

Lower bound for the REPLICA COUNTING problem

Obvious lower bound: $\left\lceil \frac{\sum_{i \in \mathcal{C}} r_i}{W} \right\rceil = 2$
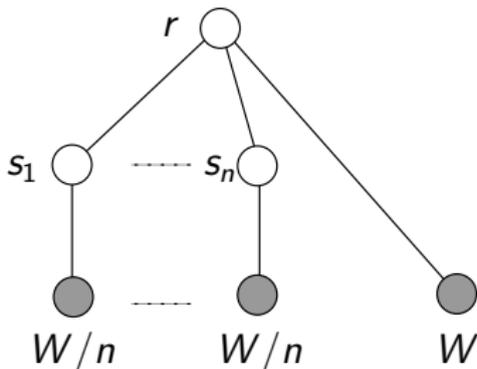
All policies require $n + 1$ replica (one at each node).

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

	REPLICA COUNTING **Homogeneous**	REPLICA COST **Heterogeneous**
Closest **Upwards** **Multiple**	polynomial [Cidon02,Liu06]	

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

	REPLICA COUNTING	REPLICA COST
	Homogeneous	**Heterogeneous**
Closest	polynomial [Cidon02,Liu06]	
Upwards		
Multiple	polynomial algorithm	

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

| | REPLICA COUNTING | REPLICA COST |
	Homogeneous	**Heterogeneous**
Closest	polynomial [Cidon02,Liu06]	
Upwards	NP-complete	
Multiple	polynomial algorithm	

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

	REPLICA COUNTING	REPLICA COST
	Homogeneous	**Heterogeneous**
Closest	polynomial [Cidon02,Liu06]	NP-complete
Upwards	NP-complete	NP-complete
Multiple	polynomial algorithm	NP-complete

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)
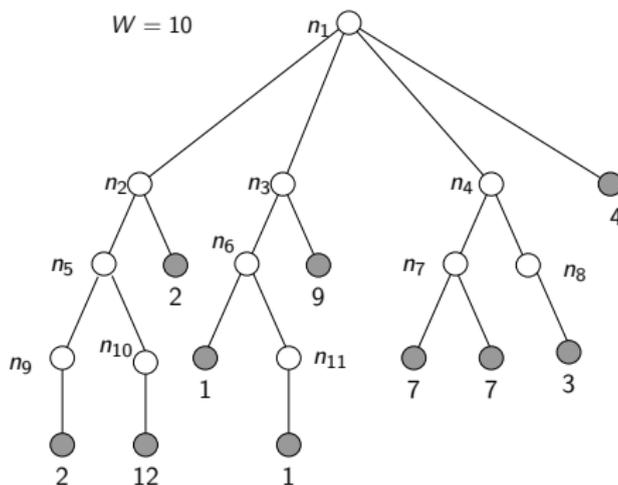
Multiple/Homogeneous: example

Initial network

The example network

Multiple/Homogeneous: example
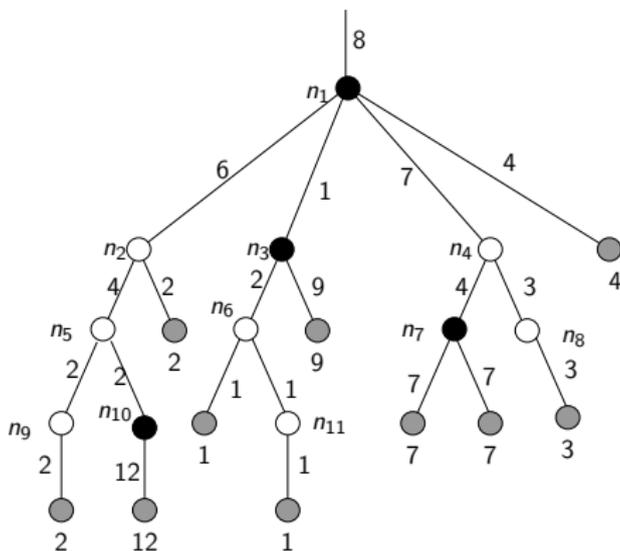
Pass 1

Placing *saturated* replicas

Multiple/Homogeneous: example
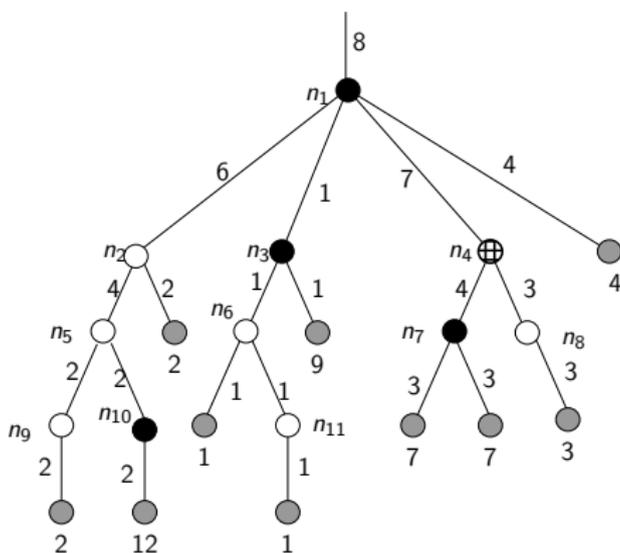
Pass 2

Placing extra replicas: n_4 has maximum useful flow

Multiple/Homogeneous: example
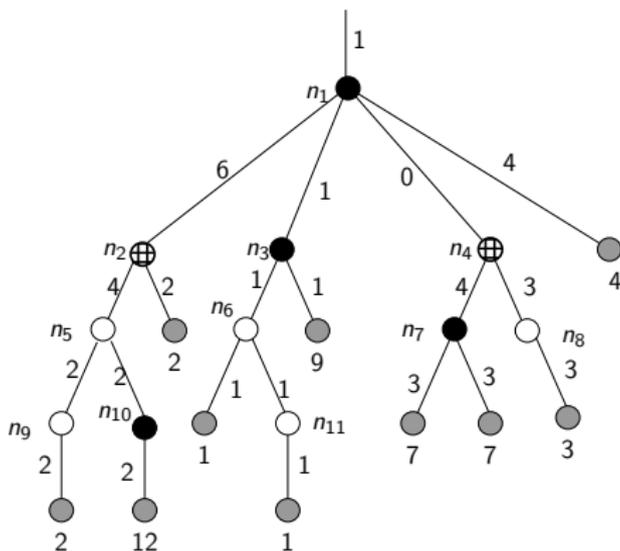
Pass 2

Placing extra replicas: n_2 is of maximum useful flow 1

Multiple/Homogeneous: example
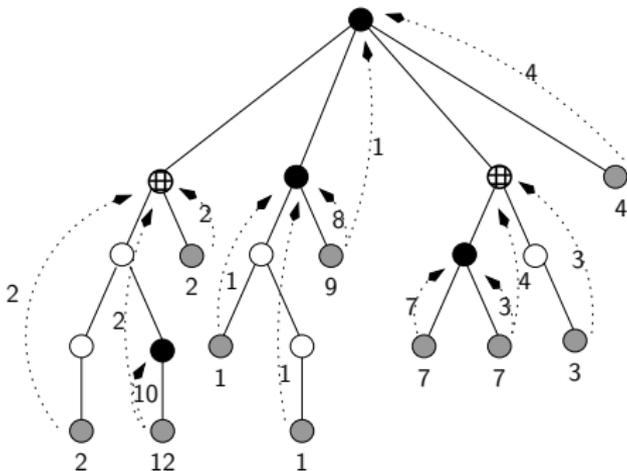
Pass 3

Deciding where requests are processed

Upwards/Homogeneous

- The REPLICA COUNTING problem with the *Upwards* strategy is NP-complete in the strong sense
- Reduction from 3-PARTITION

n_m

n_2

n_1

$W = B$

c_1 c_2 c_3 c_{3m}
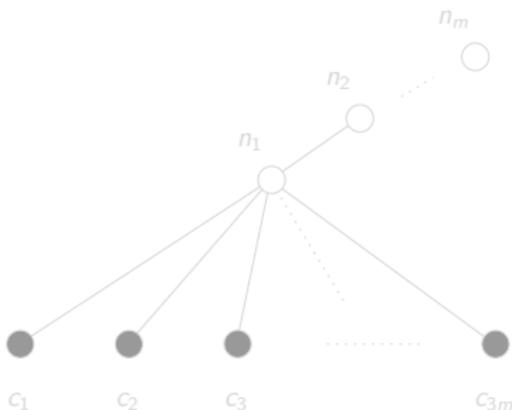
$\sum_{i=1}^{3m} a_i = mB$

Upwards/Homogeneous

- The REPLICA COUNTING problem with the *Upwards* strategy is NP-complete in the strong sense
- Reduction from 3-PARTITION
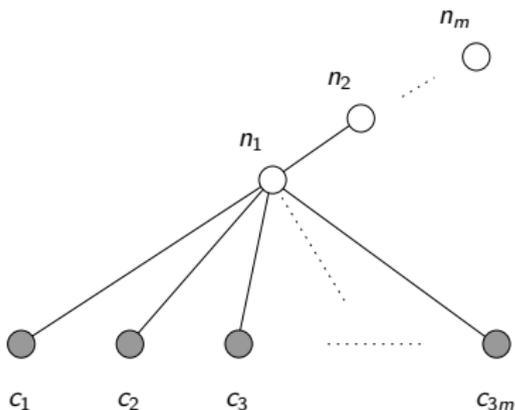
$W = B$

$\sum_{i=1}^{3m} a_i = mB$

Heterogeneous network: REPLICA COST problem

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

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

Solution with total storage cost $S + 1$?

Intro
oooo
Framework
ooooo
Policies
oooooooo
Complexity
ooooo●
LP
ooooo
Heuristics
oooo
Experiments
ooooooo
Extensions
ooo
Conclusion
ooo

Heterogeneous network: Replica Cost problem

- All three instances of the Replica Cost problem with heterogeneous nodes are NP-complete
- Reduction from 2-PARTITION
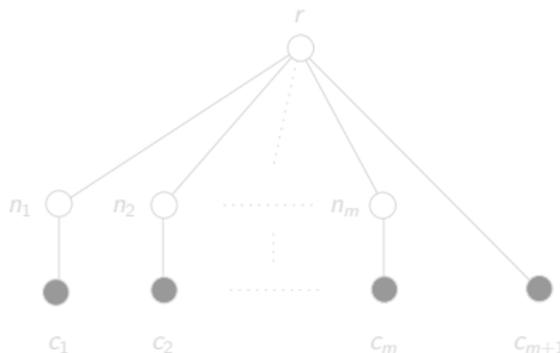
$$\sum_{i=1}^{m} a_i = S, \ a_{m+1} = 1, \ W_j = a_i, \ W_r = S/2 + 1$$

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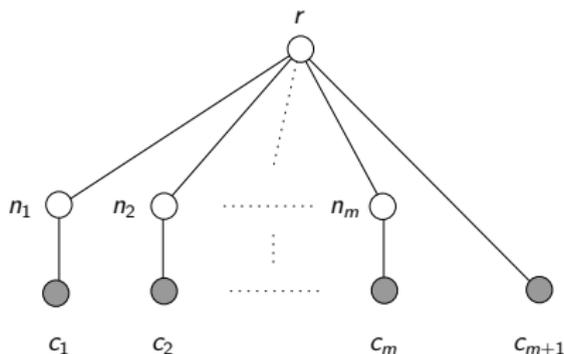
$\sum_{i=1}^{m} a_i = S$, $a_{m+1} = 1$, $W_j = a_i$, $W_r = S/2 + 1$

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 $\text{server}(i)$
    - 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 $l \in \text{path}[i \to r]$ used when i accesses $\text{server}(i)$
  - If $l \notin \text{path}[i \to r]$, $z_{i,l} = 0$

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

Objective function: $\sum_{j \in \mathcal{N}} sc_j x_j$

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 $l \in \text{path}[i \to r]$ used when i accesses $\text{server}(i)$
  - If $l \notin \text{path}[i \to r]$, $z_{i,l} = 0$

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

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**: $\forall i \in \mathcal{C}, \sum_{j \in \mathsf{Ancestors}(i)} y_{i,j} = 1$
- **Links**: $\forall i \in \mathcal{C}, z_{i,i \to \mathsf{parent}(i)} = 1$
- Conservation: $\forall i \in \mathcal{C}, \forall l : j \to j' = \mathsf{parent}(j) \in \mathsf{path}[i \to r]$,
  $$z_{i,\mathsf{succ}(l)} = z_{i,l} - y_{i,j'}$$
- 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: $\forall i \in \mathcal{C}, \forall j \in \mathsf{Ancestors}(i), \mathsf{dist}(i,j) y_{i,j} \leq q_i$

- *Closest* constraint: $\forall i \in \mathcal{C}, \forall j \in \mathsf{Ancestors}(i) \setminus \{r\}$,
  $$\forall i' \in \mathcal{C} \cap \mathsf{subtree}(j), y_{i,j} + z_{i',j \to \mathsf{parent}(j)} \leq 1$$

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: $\forall i \in \mathcal{C}, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1$
- 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: $\forall i \in \mathcal{C}, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1$
- 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: $\forall i \in \mathcal{C}, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1$
- 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**: $\forall i \in \mathcal{C}, \sum_{j \in \text{Ancestors}(i)} y_{i,j} = 1$
- **Links**: $\forall i \in \mathcal{C}, z_{i,i \to \text{parent}(i)} = 1$
- **Conservation**: $\forall i \in \mathcal{C}, \forall I : j \to j' = \text{parent}(j) \in \text{path}[i \to r],$
  $$z_{i,\text{succ}(I)} = z_{i,I} - y_{i,j'}$$
- **Server capacity**: $\forall j \in \mathcal{N}, \sum_{i \in \mathcal{C}} r_i y_{i,j} \leq W_j x_j$
- **Bandwidth limit**: $\forall I \in \mathcal{L}, \sum_{i \in \mathcal{C}} r_i z_{i,I} \leq BW_I$
- **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: $\forall i \in \mathcal{C}, \forall j \in \text{Ancestors}(i) \setminus \{r\},$
  $\forall i' \in \mathcal{C} \cap \text{subtree}(j), y_{i,j} + z_{i',j \to \text{parent}(j)} \leq 1$

Multiple formulation

Multiple

- 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 $x_j = 0.5$.

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 $x_j = 0.5$.

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

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 $s = |\mathcal{C}| + |\mathcal{N}|$ 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 $O(s^2)$,
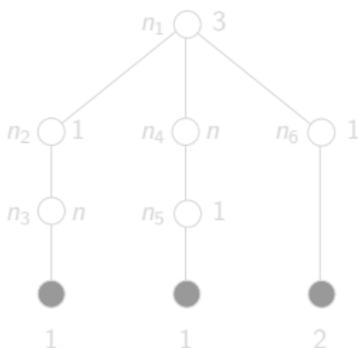 where $s = |\mathcal{C}| + |\mathcal{N}|$ is problem size

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 $s = |\mathcal{C}| + |\mathcal{N}|$ 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
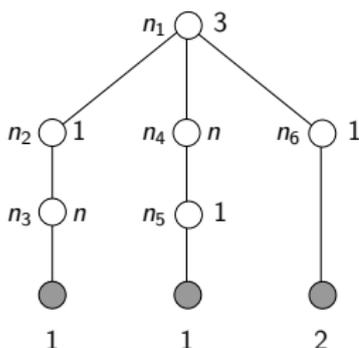 - Choosing n_2, n_4 and then n_1

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
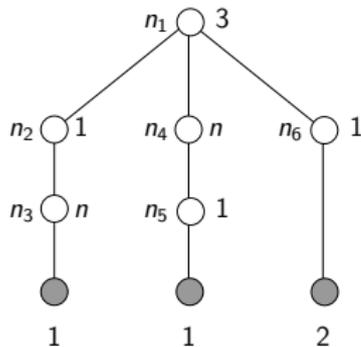   - Choosing n_2, n_4 and then n_1

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
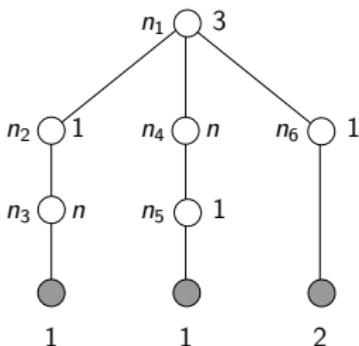
Heuristics for *Upwards*

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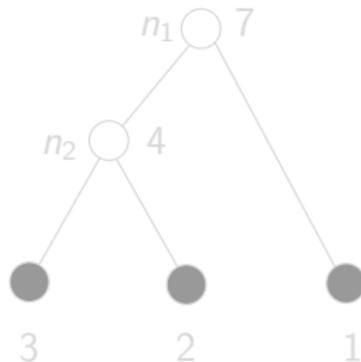
Heuristics for *Upwards*
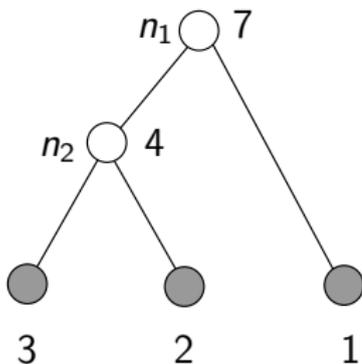
- Upwards Top Down **UTD**
  - 2-pass algorithm
  - Select first saturating nodes, then extra nodes
  - Choosing n_2 (for c_1) and in second pass n_1 (for c_2, c_3)

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

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

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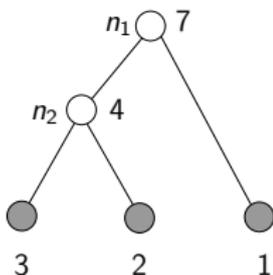
Heuristics for *Multiple*

- 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

- 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
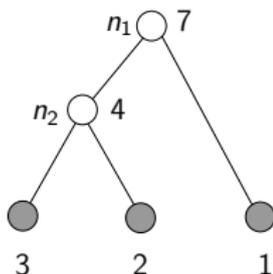
Heuristics for *Multiple*

- 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

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

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:

$$\lambda = \frac{\sum_{i \in \mathcal{C}} r_i}{\sum_{j \in \mathcal{N}} W_i}$$

- 30 trees for each $\lambda = 0.1, 0.2, ..., 0.9$
- Problem size $s = |\mathcal{C}| + |\mathcal{N}|$ 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:

$$\lambda = \frac{\sum_{i \in \mathcal{C}} r_i}{\sum_{j \in \mathcal{N}} W_i}$$

- 30 trees for each $\lambda = 0.1, 0.2, ..., 0.9$
- Problem size $s = |\mathcal{C}| + |\mathcal{N}|$ 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

Results - Percentage of success

- 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

Results - Percentage of success

- 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_λ: 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_λ: 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

- Homogeneous results
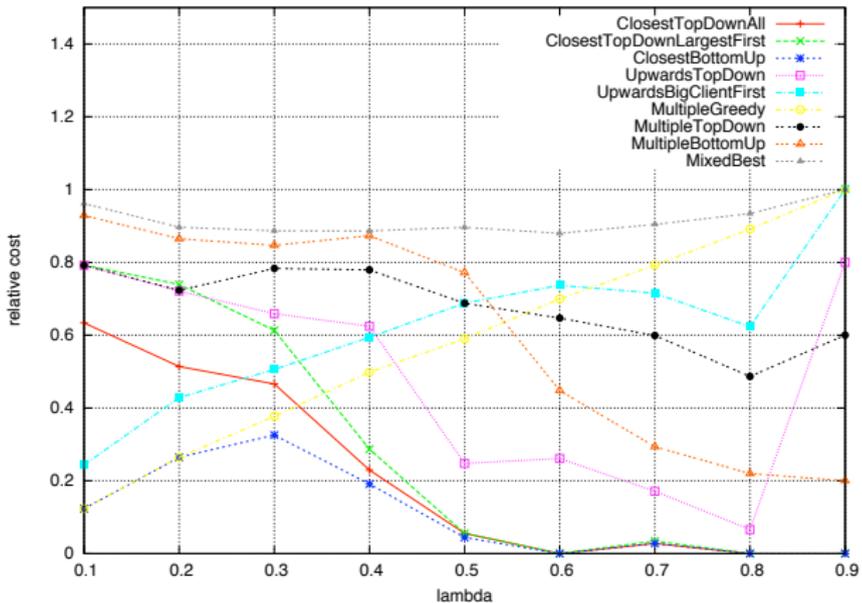
Results - Solution cost

- Heterogeneous results - similar to the homogeneous case
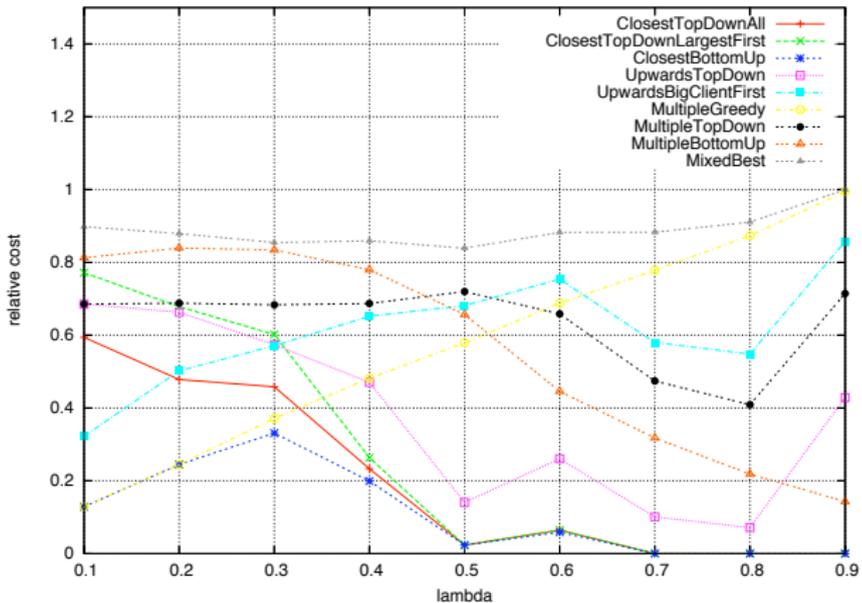
Summary

- 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

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

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

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

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

- 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

- 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
- 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

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.

Intro	Framework	Policies	Complexity	LP	Heuristics	Experiments	Extensions	Conclusion

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 ☺