Comparison of Access Policies for Replica Placement in Tree Networks

Anne Benoit

LIP, École Normale Supérieure de Lyon, France

Euro-Par 2009
August 27, 2009

Introduction and motivation

- Replica placement in tree networks
- Set of clients (tree leaves): requests 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 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 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?

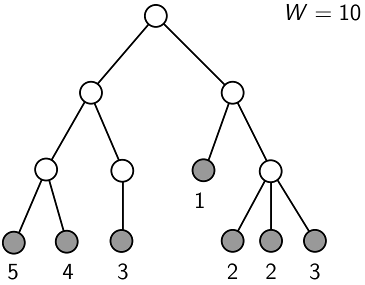
Introduction and motivation

- Replica placement in tree networks
- Set of clients (tree leaves): requests 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?

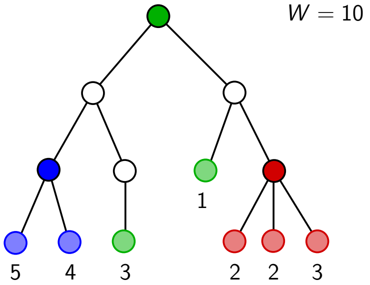
Rule of the game

- Handle all client requests, and minimize cost of replicas
- Several policies to assign replicas



Rule of the game

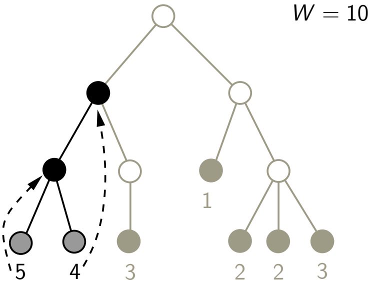
- Handle all client requests, and minimize cost of replicas
- Several policies to assign replicas



Single with *Closest*

Rule of the game

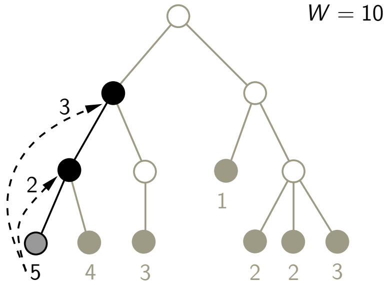
- Handle all client requests, and minimize cost of replicas
- Several policies to assign replicas



Single

Rule of the game

- Handle all client requests, and minimize cost of replicas
- Several policies to assign replicas



Multiple

Outline

Definitions and notations

- Distribution tree: clients $\mathcal{C}$ (leaf nodes), internal nodes $\mathcal{N}$
- **Client** $i \in \mathcal{C}$:
  - Sends r_i requests *per time unit*
- **Node** $j \in \mathcal{N}$:
  - Can contain the object database replica (server) or not
  - Processing capacity W_j
  - Storage cost sc_j
- **Tree notations**
  - r: tree root
  - children(j): set of children of node $j \in \mathcal{N}$
  - parent(k): parent in the tree of node $k \in \mathcal{N} \cup \mathcal{C}$
  - ancestors(k): set of ancestors of node k
  - subtree(k): subtree rooted in k, including k

Definitions and notations

- Distribution tree: clients $\mathcal{C}$ (leaf nodes), internal nodes $\mathcal{N}$
- **Client** $i \in \mathcal{C}$:
  - Sends r_i requests *per time unit*
- **Node** $j \in \mathcal{N}$:
  - Can contain the object database replica (server) or not
  - Processing capacity W_j
  - Storage cost sc_j
- **Tree notations**
  - r: tree root
  - children(j): set of children of node $j \in \mathcal{N}$
  - parent(k): parent in the tree of node $k \in \mathcal{N} \cup \mathcal{C}$
  - ancestors(k): set of ancestors of node k
  - subtree(k): subtree rooted in k, including k

Definitions and notations

- Distribution tree: clients $\mathcal{C}$ (leaf nodes), internal nodes $\mathcal{N}$
- **Client** $i \in \mathcal{C}$:
  - Sends r_i requests *per time unit*
- **Node** $j \in \mathcal{N}$:
  - Can contain the object database replica (server) or not
  - Processing capacity W_j
  - Storage cost sc_j
- **Tree notations**
  - r: tree root
  - children(j): set of children of node $j \in \mathcal{N}$
  - parent(k): parent in the tree of node $k \in \mathcal{N} \cup \mathcal{C}$
  - ancestors(k): set of ancestors of node k
  - subtree(k): subtree rooted in k, including k

Definitions and notations

- Distribution tree: clients $\mathcal{C}$ (leaf nodes), internal nodes $\mathcal{N}$
- **Client** $i \in \mathcal{C}$:
  - Sends r_i requests *per time unit*
- **Node** $j \in \mathcal{N}$:
  - Can contain the object database replica (server) or not
  - Processing capacity W_j
  - Storage cost sc_j
- **Tree notations**
  - r: tree root
  - children(j): set of children of node $j \in \mathcal{N}$
  - parent(k): parent in the tree of node $k \in \mathcal{N} \cup \mathcal{C}$
  - ancestors(k): set of ancestors of node k
  - subtree(k): subtree rooted in k, including k

Problem definition

- Goal: place replicas to process client requests
- Client $i \in \mathcal{C}$: servers(i) $\subseteq$ ancestors(i) 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
- Server capacity constraint: $\forall s \in R, \sum_{i \in \mathcal{C} | s \in \text{servers}(i)} r_{i,s} \leq \text{W}_s$
- Objective function: Min $\sum_{s \in R} \text{sc}_s$

Problem definition

- Goal: place replicas to process client requests
- Client $i \in \mathcal{C}$: servers(i) $\subseteq$ ancestors(i) set of servers responsible for processing its requests
- $r_{i,s}$: number of requests from client i processed by server s ($\sum_{s \in \text{servers}(i)} r_{i,s} = r_i$)
- $R = \{s \in \mathcal{N} \mid \exists i \in \mathcal{C}\ ,\ s \in \text{servers}(i)\}$: set of replicas
- Server capacity constraint: $\forall s \in R, \sum_{i \in \mathcal{C} | s \in \text{servers}(i)} r_{i,s} \leq \text{W}_s$
- Objective function: Min $\sum_{s \in R} \text{sc}_s$

Problem definition

- Goal: place replicas to process client requests
- Client $i \in \mathcal{C}$: servers(i) $\subseteq$ ancestors(i) 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
- Server capacity constraint: $\forall s \in R, \sum_{i \in \mathcal{C} | s \in \text{servers}(i)} r_{i,s} \leq \mathsf{W}_s$
- Objective function: Min $\sum_{s \in R} \mathsf{sc}_s$

Problem instances

- Number of servers assigned to each client:

  Single. A unique server handles the r_i requests of client i ($|\text{servers}(i)| = 1$)

  Multiple. Several servers in the set servers(i)

- Platform types:

  Different servers. Restrict to case where $\text{sc}_s = W_s$, Replica Cost problem

  Identical servers. Identical node capacities ($\forall s \in \mathcal{N}$, $\text{W}_s = \text{W}$), $\text{sc}_s = 1$, Replica Counting problem

- Literature: *Single* further constrained to *Closest* (server of client i: first server on the path from i to r)

Problem instances

- Number of servers assigned to each client:

  Single. A unique server handles the r_i requests of client i ($|\text{servers}(i)| = 1$)

  Multiple. Several servers in the set servers(i)

- Platform types:

  Different servers. Restrict to case where $\text{sc}_s = W_s$, Replica Cost problem

  Identical servers. Identical node capacities ($\forall s \in \mathcal{N},\ \text{W}_s = \text{W}$), $\text{sc}_s = 1$, Replica Counting problem

- Literature: *Single* further constrained to *Closest* (server of client i: first server on the path from i to r)

Problem instances

- Number of servers assigned to each client:

  Single. A unique server handles the r_i requests of client i ($|\text{servers}(i)| = 1$)

  Multiple. Several servers in the set servers(i)

- Platform types:

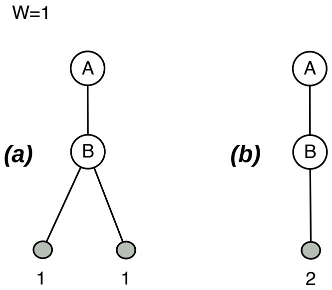
  Different servers. Restrict to case where $\text{sc}_s = W_s$, Replica Cost problem

  Identical servers. Identical node capacities ($\forall s \in \mathcal{N},\ \mathrm{W}_s = \mathrm{W}$), $\text{sc}_s = 1$, Replica Counting problem

- Literature: *Single* further constrained to *Closest* (server of client i: first server on the path from i to r)

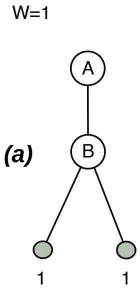
Outline

Solution existence



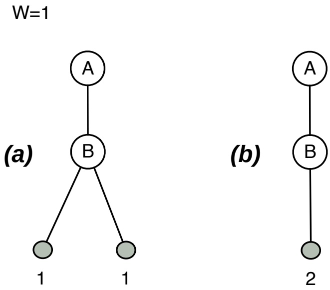
- (a): One replica per node, solution for *Single* (and thus for *Multiple*)
- (b): Solution only with *Multiple*

Solution existence



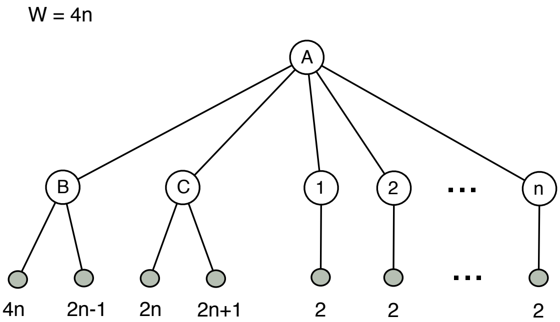
- (a): One replica per node, solution for *Single* (and thus for *Multiple*)
- (b): Solution only with *Multiple*

Solution existence



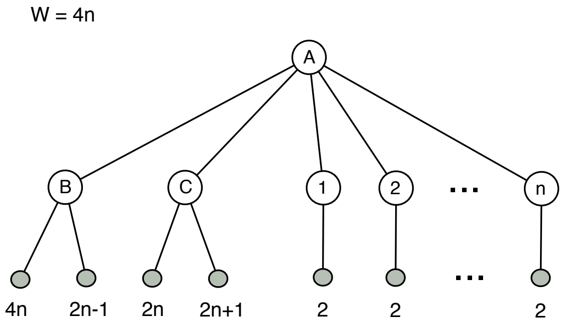
- (a): One replica per node, solution for *Single* (and thus for *Multiple*)
- (b): Solution only with *Multiple*

Solution cost for Replica Counting



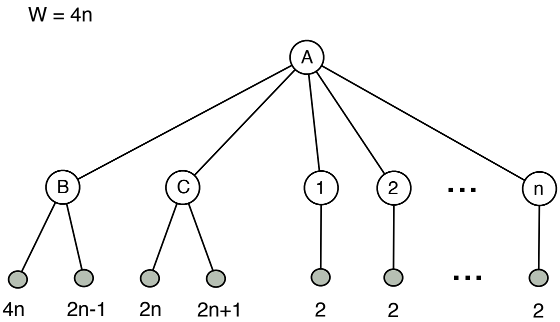
- *Multiple*: 3 replicas in A, B and C
- *Single*: replicas everywhere ($n+3$)
- Performance factor: $\frac{n+3}{3}$, can be arbitrarily big

Solution cost for REPLICA COUNTING



- *Multiple*: 3 replicas in A, B and C
- *Single*: replicas everywhere $(n+3)$
- Performance factor: $\frac{n+3}{3}$, can be arbitrarily big

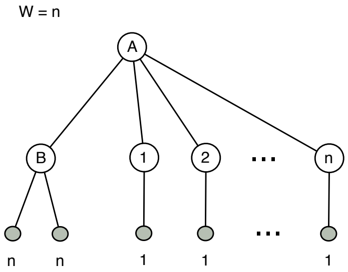
Solution cost for REPLICA COUNTING



- *Multiple*: 3 replicas in A, B and C
- *Single*: replicas everywhere $(n+3)$
- Performance factor: $\frac{n+3}{3}$, can be arbitrarily big

Lower bound for REPLICA COUNTING

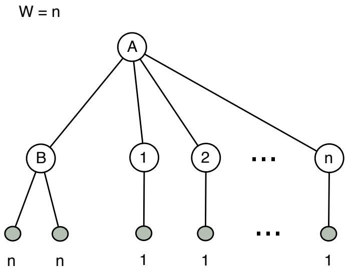
Obvious lower bound: $\left\lceil \frac{\sum_{i \in \mathcal{C}} r_i}{W} \right\rceil = 3n/n = 3$



All policies require $n + 2$ replica (one at each node).

Lower bound for Replica Counting

Obvious lower bound: $\left\lceil \frac{\sum_{i \in \mathcal{C}} r_i}{W} \right\rceil = 3n/n = 3$



All policies require $n+2$ replica (one at each node).

Outline

1. Framework
2. Access policies comparison
3. Complexity results
4. From Multiple to Single
5. Conclusion

Complexity results

	Single	*Multiple*
Replica Counting	NP-complete	polynomial
Replica Cost	NP-complete	NP-complete

- *Single*/Replica Counting: NP-hard, while polynomial with *Closest* (dynamic programming algorithms)
- Replica Cost: NP-hard because of resource heterogeneity
- *Multiple*/Replica Counting: only polynomial case, multi-pass greedy algorithm
- *(Proofs: see TPDS'2008 paper 19(12), 1614-1627)*

Complexity results

	Single	*Multiple*
Replica Counting	NP-complete	polynomial
Replica Cost	NP-complete	NP-complete

- *Single*/Replica Counting: NP-hard, while polynomial with *Closest* (dynamic programming algorithms)
- Replica Cost: NP-hard because of resource heterogeneity
- *Multiple*/Replica Counting: only polynomial case, multi-pass greedy algorithm
- *(Proofs: see TPDS'2008 paper 19(12), 1614-1627)*

Complexity results

	Single	*Multiple*
REPLICA COUNTING	NP-complete	polynomial
REPLICA COST	NP-complete	NP-complete

- *Single*/REPLICA COUNTING: NP-hard, while polynomial with *Closest* (dynamic programming algorithms)
- REPLICA COST: NP-hard because of resource heterogeneity
- *Multiple*/REPLICA COUNTING: only polynomial case, multi-pass greedy algorithm
- *(Proofs: see TPDS'2008 paper 19(12), 1614-1627)*

Complexity results

	Single	*Multiple*
Replica Counting	NP-complete	polynomial
Replica Cost	NP-complete	NP-complete

- *Single*/Replica Counting: NP-hard, while polynomial with *Closest* (dynamic programming algorithms)
- Replica Cost: NP-hard because of resource heterogeneity
- *Multiple*/Replica Counting: only polynomial case, multi-pass greedy algorithm
- *(Proofs: see TPDS'2008 paper 19(12), 1614-1627)*

Outline

Problem formulation

- Procedure to build a *Single* allocation for REPLICA COUNTING with a guarantee on the cost
- *Single* can be arbitrarily worse than *Multiple*: when can we derive good *Single* solutions?

Let $(\mathcal{C}, \mathcal{N})$ be a problem instance in which $r_i \leq W$ for all $i \in \mathcal{C}$ (otherwise, there is no solution to the *Single* problem).
We are given an optimal *Multiple* solution for this problem, of cost M (i.e., M is the number of servers in this solution).
We aim at finding a *Single* solution with a cost $S \leq 2M$, and at characterizing cases in which this is possible.

Problem formulation

- Procedure to build a *Single* allocation for REPLICA COUNTING with a guarantee on the cost
- *Single* can be arbitrarily worse than *Multiple*: when can we derive good *Single* solutions?

Let $(\mathcal{C}, \mathcal{N})$ be a problem instance in which $r_i \leq W$ for all $i \in \mathcal{C}$ (otherwise, there is no solution to the *Single* problem).
We are given an optimal *Multiple* solution for this problem, of cost M (i.e., M is the number of servers in this solution).
We aim at finding a *Single* solution with a cost $S \leq 2M$, and at characterizing cases in which this is possible.

Problem formulation

- Procedure to build a *Single* allocation for REPLICA COUNTING with a guarantee on the cost
- *Single* can be arbitrarily worse than *Multiple*: when can we derive good *Single* solutions?

Let $(\mathcal{C}, \mathcal{N})$ be a problem instance in which $r_i \leq W$ for all $i \in \mathcal{C}$ (otherwise, there is no solution to the *Single* problem).
We are given an optimal *Multiple* solution for this problem, of cost M (i.e., M is the number of servers in this solution).
We aim at finding a *Single* solution with a cost $S \leq 2M$, and at characterizing cases in which this is possible.

Linear trees

- $|\mathcal{N}| = n$ nodes rooted in node 1: $1 \rightarrow 2 \rightarrow \cdots \rightarrow n$
- $\mathcal{C}_j$: set of clients attached to node j
- Condition:

$$jW \geq 2 \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i$$

  At each tree level, twice more nodes than min nb of servers requested to handle all requests from root to this level
- Condition on the whole tree: $n \geq \frac{2}{W} \sum_{i \in \mathcal{C}} r_i$
- Procedure which assigns servers, never fails because of condition

Linear trees

- $|\mathcal{N}| = n$ nodes rooted in node 1: $1 \to 2 \to \cdots \to n$
- $\mathcal{C}_j$: set of clients attached to node j
- Condition:

$$jW \geq 2 \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i$$

At each tree level, twice more nodes than min nb of servers requested to handle all requests from root to this level

- Condition on the whole tree: $n \geq \frac{2}{W} \sum_{i \in \mathcal{C}} r_i$
- Procedure which assigns servers, never fails because of condition

Linear trees

- $|\mathcal{N}| = n$ nodes rooted in node 1: $1 \rightarrow 2 \rightarrow \cdots \rightarrow n$
- $\mathcal{C}_j$: set of clients attached to node j
- Condition:

$$jW \geq 2 \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i$$

At each tree level, twice more nodes than min nb of servers requested to handle all requests from root to this level

- Condition on the whole tree: $n \geq \frac{2}{W} \sum_{i \in \mathcal{C}} r_i$
- Procedure which assigns servers, never fails because of condition

linear-tree procedure

procedure linear-tree $(\mathcal{C}, \mathcal{N})$

$\forall i \in \mathcal{C}, \forall j \in \mathcal{N}, s_{i,j} = 0;\ \ \forall j \in \mathcal{N}, s_j = 0;$ *// Initialisation.*

for $j = 1..n$ **do** **for** $i \in \mathcal{C}_j$ **do**

// Loop 1: try to add requests to an existing server.

for $j' = j..1$ **do**

  if $\sum_{k \in \mathcal{N}} s_{i,k} = 0$ *and* $s_{j'} \neq 0$ **then**

    if $r_i + s_{j'} \leq W$ **then** $s_{i,j'} = r_i;\ s_{j'} = s_{j'} + r_i$

  end

end

// Loop 2: If Loop 1 did not succeed, create a new server.

if $\sum_{k \in \mathcal{N}} s_{i,k} = 0$ **then**

  for $j' = j..1$ **do** **if** $s_{j'} = 0$ **then** $s_{i,j'} = r_i;\ s_{j'} = r_i;$ break;

end

return $\{s_{i,j} \mid i \in \mathcal{C}, 1 \leq j \leq n\}$

Linear trees: proof of correctness

- S_j: nb of servers allocated at each step of loop on j
- We prove that $S_j \leq j$: enough nodes available, Loop 2 always find a node with no requests
- Note that $s_k + s_{k'} > W$ for all (k, k') (greedy allocation), all servers but the last one handle at least $W/2$ requests
- If $S_j = 1$, then $S_j \leq j$ since $j \geq 1$
- If $S_j \geq 2$, $s_k + s_{k'} > W$ and other servers with at least $W/2$ requests, thus $req > (S_j - 2)\frac{W}{2} + W = S_j\frac{W}{2}$. We have $req = \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i$, thus, with the condition,

$$S_j < \frac{2}{W} \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i \leq j$$

Linear trees: proof of correctness

- S_j: nb of servers allocated at each step of loop on j
- We prove that $S_j \leq j$: enough nodes available, Loop 2 always find a node with no requests
- Note that $s_k + s_{k'} > W$ for all (k, k') (greedy allocation), all servers but the last one handle at least $W/2$ requests
- If $S_j = 1$, then $S_j \leq j$ since $j \geq 1$
- If $S_j \geq 2$, $s_k + s_{k'} > W$ and other servers with at least $W/2$ requests, thus $req > (S_j - 2)\frac{W}{2} + W = S_j\frac{W}{2}$. We have $req = \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i$, thus, with the condition,

$$S_j < \frac{2}{W} \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i \leq j$$

Linear trees: proof of correctness

- S_j: nb of servers allocated at each step of loop on j
- We prove that $S_j \leq j$: enough nodes available, Loop 2 always find a node with no requests

- Note that $s_k + s_{k'} > W$ for all (k, k') (greedy allocation), all servers but the last one handle at least $W/2$ requests
- If $S_j = 1$, then $S_j \leq j$ since $j \geq 1$
- If $S_j \geq 2$, $s_k + s_{k'} > W$ and other servers with at least $W/2$ requests, thus $req > (S_j - 2)\frac{W}{2} + W = S_j \frac{W}{2}$. We have $req = \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i$, thus, with the condition,

$$S_j < \frac{2}{W} \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i \leq j$$

Linear trees: proof of correctness

- S_j: nb of servers allocated at each step of loop on j
- We prove that $S_j \leq j$: enough nodes available, Loop 2 always find a node with no requests

- Note that $s_k + s_{k'} > W$ for all (k, k') (greedy allocation), all servers but the last one handle at least $W/2$ requests
- If $S_j = 1$, then $S_j \leq j$ since $j \geq 1$
- If $S_j \geq 2$, $s_k + s_{k'} > W$ and other servers with at least $W/2$ requests, thus $req > (S_j - 2)\frac{W}{2} + W = S_j \frac{W}{2}$. We have $req = \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i$, thus, with the condition,

$$S_j < \frac{2}{W} \sum_{i \in \cup_{1 \leq k \leq j} \mathcal{C}_k} r_i \leq j$$

Linear trees: proof of cost

- Multiple solution handles all requests:

$$M \geq \left\lceil \frac{1}{W} \sum_{i \in \mathcal{C}} r_i \right\rceil$$

- Number of servers in the new solution:

$$S = S_n \leq \frac{2}{W} \sum_{i \in \mathcal{C}} r_i \leq 2M$$

Linear trees: proof of cost

- Multiple solution handles all requests:

$$M \geq \left\lceil \frac{1}{W} \sum_{i \in \mathcal{C}} r_i \right\rceil$$

- Number of servers in the new solution:

$$S = S_n \leq \frac{2}{W} \sum_{i \in \mathcal{C}} r_i \leq 2M$$

General trees

- Problem: several branches of the tree in conflict
- Solution: apply **linear-tree** procedure on each tree branch, but need a condition on the min nb of nodes on each branch
- New constraint:

$$\forall j \in \{j' \in \mathcal{N} \mid |\text{children}(j') \cap \mathcal{N}| \geq 2\} \cup \{r\},\ \forall k \in \text{subtree}(j) \cap \mathcal{N},$$
$$\text{let } X = \{j\} \cup \text{ancestors}(k) \cap \text{subtree}(j).$$
$$\text{Then } |X| \geq \frac{2}{W} \sum_{\ell \in X} \sum_{i \in \text{children}(\ell) \cap \mathcal{C}} r_i \quad (1)$$

general-tree procedure

procedure general-tree $(\mathcal{C}, \mathcal{N})$

1. $\forall j \in \mathcal{N},\ br(j) = |\text{children}(j) \cap \mathcal{N}|$ (nb of branches rooted in j not yet processed)
2. Call **linear-tree** on leftmost branch, current branch $cb = (1, 2, ..., k) \subseteq \mathcal{N}$; cb processed: $\forall \ell \in cb,\ br(\ell) = br(\ell) - 1$
3. For $j = \max_{j' \in cb}\{j' \mid br(j') \geq 1\}$, call **linear-tree** on $cb = (j, j_1, ..., j_k)$; cb processed
4. If required, **merge-servers** on current branch
5. Go back to step 3 until $\forall j \in \mathcal{N},\ br(j) \leq 0$

General trees: proof

- Cost $S \leq 2M$ with in some cases extra constraint of binary tree
- End of step 2: at most one server handling less than $W/2$ requests, allocation possible because of constraint (1) on r
- Step 3: at most W requests attached to j_1; possible to create server at this node, idem for nodes j_2 to j_k
- End of this call: may have two servers handling less than $W/2$ requests: x and y
- Procedure **merge-servers**: aims at suppressing one of these servers

General trees: proof

- Cost $S \leq 2M$ with in some cases extra constraint of binary tree
- End of step 2: at most one server handling less than $W/2$ requests, allocation possible because of constraint (1) on r
- Step 3: at most W requests attached to j_1; possible to create server at this node, idem for nodes j_2 to j_k
- End of this call: may have two servers handling less than $W/2$ requests: x and y
- Procedure **merge-servers**: aims at suppressing one of these servers

General trees: proof

- Cost $S \leq 2M$ with in some cases extra constraint of binary tree
- End of step 2: at most one server handling less than $W/2$ requests, allocation possible because of constraint (1) on r
- Step 3: at most W requests attached to j_1; possible to create server at this node, idem for nodes j_2 to j_k
- End of this call: may have two servers handling less than $W/2$ requests: x and y
- Procedure **merge-servers**: aims at suppressing one of these servers

General trees: proof

- Cost $S \leq 2M$ with in some cases extra constraint of binary tree
- End of step 2: at most one server handling less than $W/2$ requests, allocation possible because of constraint (1) on r
- Step 3: at most W requests attached to j_1; possible to create server at this node, idem for nodes j_2 to j_k
- End of this call: may have two servers handling less than $W/2$ requests: x and y
- Procedure **merge-servers**: aims at suppressing one of these servers

General trees: proof

- Cost $S \leq 2M$ with in some cases extra constraint of binary tree
- End of step 2: at most one server handling less than $W/2$ requests, allocation possible because of constraint (1) on r
- Step 3: at most W requests attached to j_1; possible to create server at this node, idem for nodes j_2 to j_k
- End of this call: may have two servers handling less than $W/2$ requests: x and y
- Procedure **merge-servers**: aims at suppressing one of these servers

General trees: merge-servers(x,y)

- j: root of the current branch
- If $x \in$ ancestors(j), move requests processed by y to x
- If one node in ancestors(j) not yet a server, move requests of x and y onto this node
- Otherwise, thanks to constraint (1) at node j, process requests of current branch without using j
- Extra constraint: j has no more than 2 children
- Move requests from x to j (possible since j has less than $W/2$ clients), and then if necessary from y to j

General trees: merge-servers(x,y)

- j: root of the current branch
- If $x \in \text{ancestors}(j)$, move requests processed by y to x
- If one node in ancestors(j) not yet a server, move requests of x and y onto this node
- Otherwise, thanks to constraint (1) at node j, process requests of current branch without using j
- Extra constraint: j has no more than 2 children
- Move requests from x to j (possible since j has less than $W/2$ clients), and then if necessary from y to j

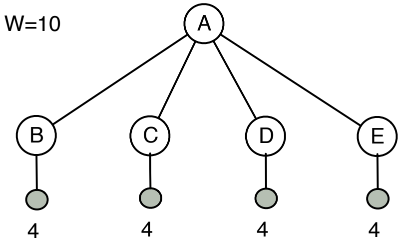
General trees: merge-servers(x,y)

- j: root of the current branch
- If $x \in$ ancestors(j), move requests processed by y to x
- If one node in ancestors(j) not yet a server, move requests of x and y onto this node
- Otherwise, thanks to constraint (1) at node j, process requests of current branch without using j
- Extra constraint: j has no more than 2 children
- Move requests from x to j (possible since j has less than $W/2$ clients), and then if necessary from y to j

General trees: merge-servers(x,y)

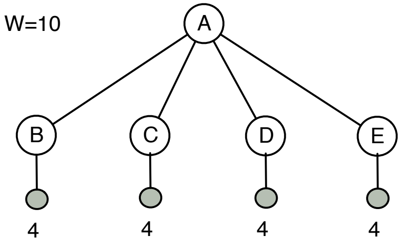
- j: root of the current branch
- If $x \in$ ancestors(j), move requests processed by y to x
- If one node in ancestors(j) not yet a server, move requests of x and y onto this node
- Otherwise, thanks to constraint (1) at node j, process requests of current branch without using j
- Extra constraint: j has no more than 2 children
- Move requests from x to j (possible since j has less than $W/2$ clients), and then if necessary from y to j

General trees: binary tree constraint



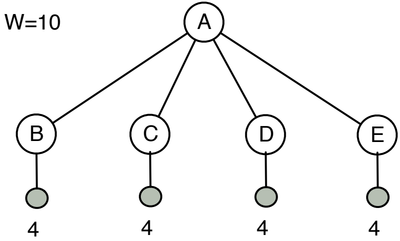
- 2 clients processed by A, and 2 servers processing $4 < W/2$ requests each
- Not possible to merge
- Performance guarantee respected since $S = M$ (3 servers requested for both policies)

General trees: binary tree constraint



- 2 clients processed by A, and 2 servers processing $4 < W/2$ requests each
- Not possible to merge
- Performance guarantee respected since $S = M$ (3 servers requested for both policies)

General trees: binary tree constraint



- 2 clients processed by A, and 2 servers processing $4 < W/2$ requests each
- Not possible to merge
- Performance guarantee respected since $S = M$ (3 servers requested for both policies)

Outline

Conclusion

- Analysis of different strategies for replica placement
- *Multiple* solution may be arbitrarily better than *Single* one
- Algorithm to build *Single* solution guaranteed to use no more than two times more servers than optimal *Multiple* solution, given constraints on problem instance
- Interesting since *Single* problem is NP-hard, and some applications may not support multiple servers

- Restrictive constraints but procedure can be applied on any tree, without guarantee
- *Intuition*: ratio of 2 should be achievable in most practical situations (to be investigated)
- Other research direction: *dynamic setting*

Conclusion

- Analysis of different strategies for replica placement
- *Multiple* solution may be arbitrarily better than *Single* one
- Algorithm to build *Single* solution guaranteed to use no more than two times more servers than optimal *Multiple* solution, given constraints on problem instance
- Interesting since *Single* problem is NP-hard, and some applications may not support multiple servers

- Restrictive constraints but procedure can be applied on any tree, without guarantee
- *Intuition*: ratio of 2 should be achievable in most practical situations (to be investigated)
- Other research direction: *dynamic setting*