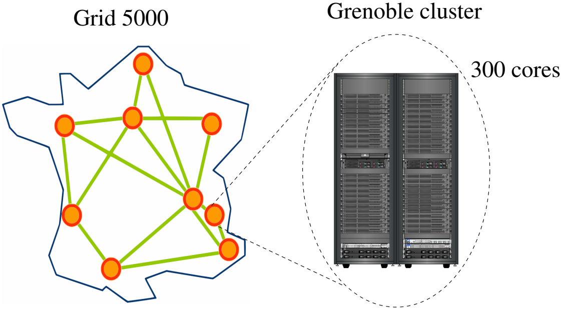
Perfect Sampling of Load Sharing Policies in Large Scale Distributed Systems

Gaël Gorgo and Jean-Marc Vincent

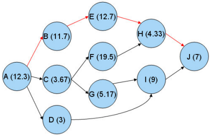
MESCAL-INRIA Project
Laboratoire d'Informatique de Grenoble
Universities of Grenoble, France
Gael.Gorgo@imag.fr, Jean-Marc.Vincent@imag.fr

Cardiff
Jun 15 2010

Large scale computing



Computation model



Load sharing middleware

- Distributed control algorithm
- migration of tasks between nodes

Practical needs

Load sharing

A controler (local) checks the utilization of the node and decides to share some work with other nodes.

- When ? $\longrightarrow$ Control triggering
- Who ? $\longrightarrow$ Paradigm Push, Pull
- Decide ? $\longrightarrow$ Local state condition
- How many ? $\longrightarrow$ Amount of work to be transfered
- Where ? $\longrightarrow$ Selection among targets (probing scheme)

Users requirements

- Maximize the utilization of resources (number of active nodes)
- Minimize the network utilization (number of transfers, costly transfers)

$\Rightarrow$ Need of a tool to evaluate the load sharing policies

Performance evaluation of Load sharing systems

Methodology

1. Quantification of the system : **steady-state evaluation**
2. Comparison of systems, paradigms, policies
3. Tuning of system parameters

Numerical approaches

- Markovian modelling and direct numerical solving
- Matrix geometric solution [ELZ86, MTS90]
- Mean field [Mit98, BGY98]
- Simulation [KH02, DKL98]

Key challenge : very large state space (C^K)

Steady-state simulation of Markov models

Generate typical state, i.e. distributed according to the steady-state

forward simulation

Run from an initial state and stop after a sufficiently long period
$\Rightarrow$ Choice of a stopping rule

Perfect simulation [PW96]

Coupling from the past scheme

- Exact stopping criteria
- Unbiased sampling
- **Monotonicity implies simulation efficiency**

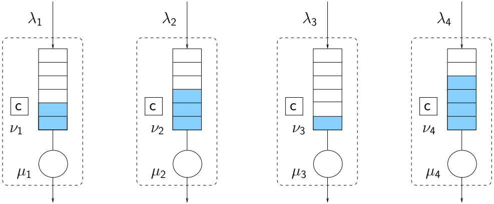
Are the load sharing systems monotone so that we can simulate them efficiently ?

Outline

Outline

Load sharing model

Parallel independent queues



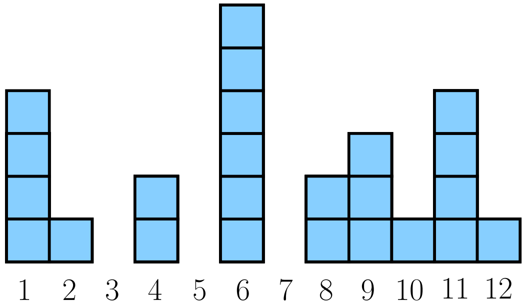
State space : number of tasks in each queue ; $\mathcal{X}_1 \times \cdots \times \mathcal{X}_K$
Dynamics : events driven by Poisson process (Poisson system [Bre99]) :

- Generation of a new task in a queue, with rate λ
- Task completion, with rate μ
- Control, with rate ν

Uniformization $\Rightarrow$ Stochastic Recurence Equation $X_{n+1} = \Phi(X_n, E_{n+1})$

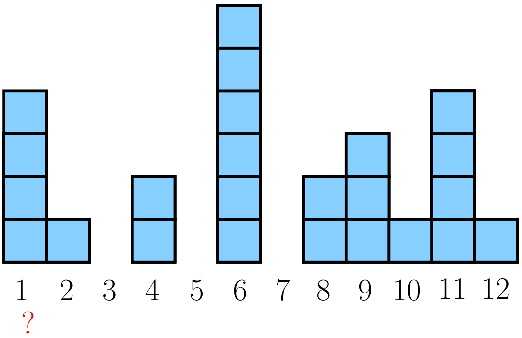
Control event example

Push to the least loaded node among potential targets (Push to the Shortest Queue)



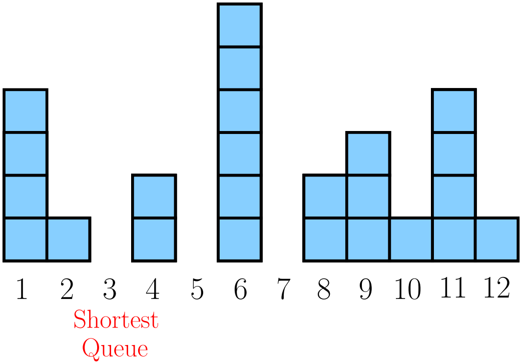
Control event example

Push to the least loaded node among potential targets (Push to the Shortest Queue)



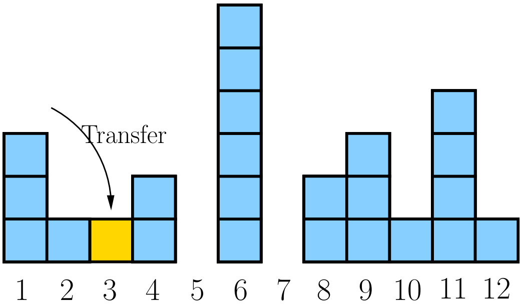
Control event example

Push to the least loaded node among potential targets (Push to the Shortest Queue)



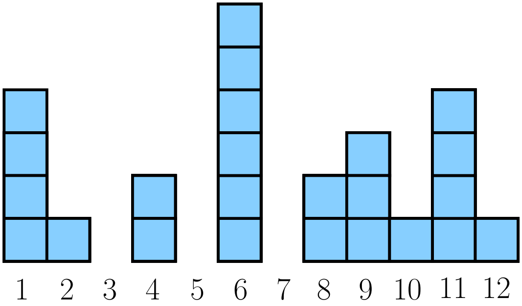
Control event example

Push to the least loaded node among potential targets (Push to the Shortest Queue)



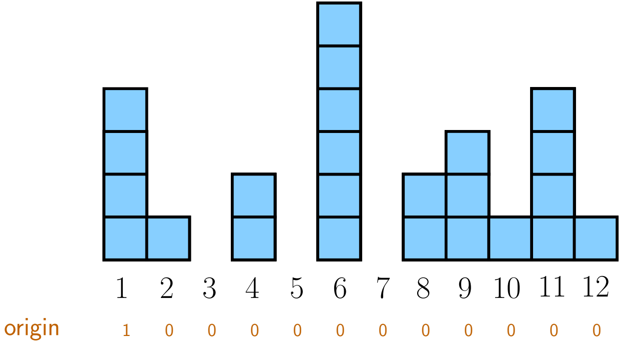
Corresponding index model

Push to the least loaded node among potential targets (Push to the Shortest Queue)



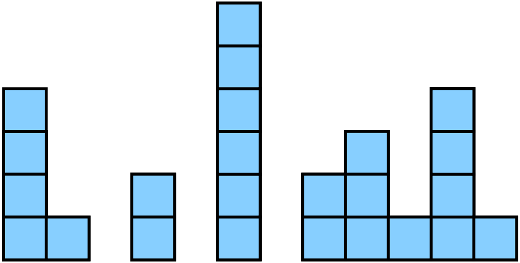
Corresponding index model

Push to the least loaded node among potential targets (Push to the Shortest Queue)



Corresponding index model

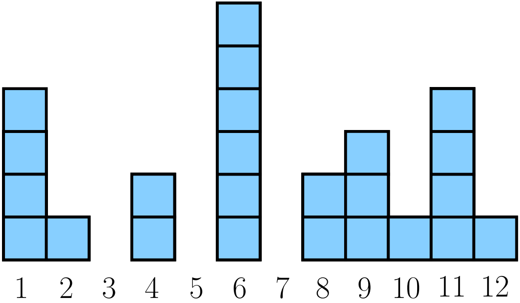
Push to the least loaded node among potential targets (Push to the Shortest Queue)



	1	2	3	4	5	6	7	8	9	10	11	12
origin	1	0	0	0	0	0	0	0	0	0	0	0
target	4	1	0	2	0	6	0	2	3	1	4	1

Corresponding index model

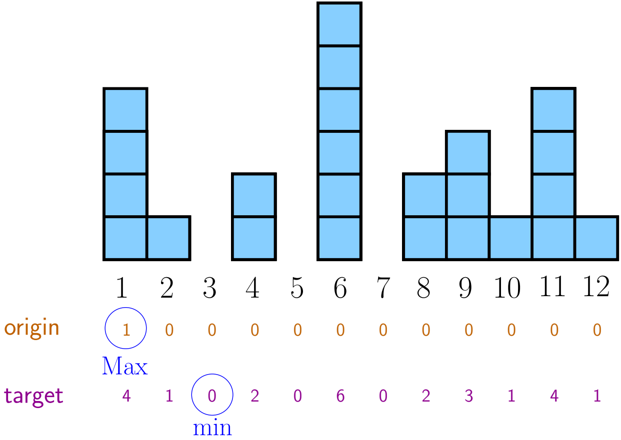
Push to the least loaded node among potential targets (Push to the Shortest Queue)



origin	1	0	0	0	0	0	0	0	0	0	0	0
	Max											
target	4	1	0	2	0	6	0	2	3	1	4	1

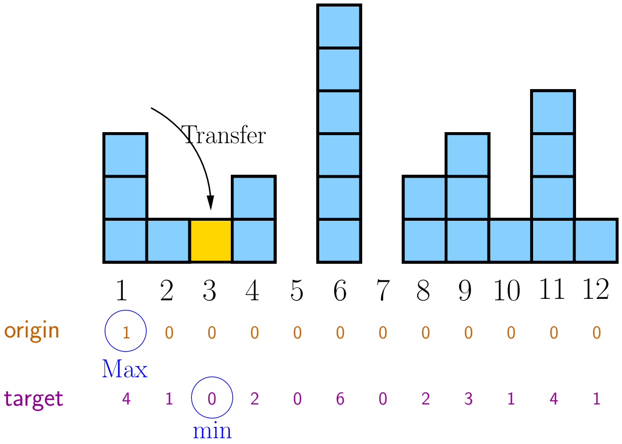
Corresponding index model

Push to the least loaded node among potential targets (Push to the Shortest Queue)



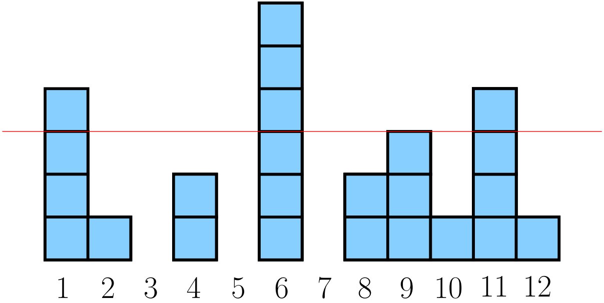
Corresponding index model

Push to the least loaded node among potential targets (Push to the Shortest Queue)



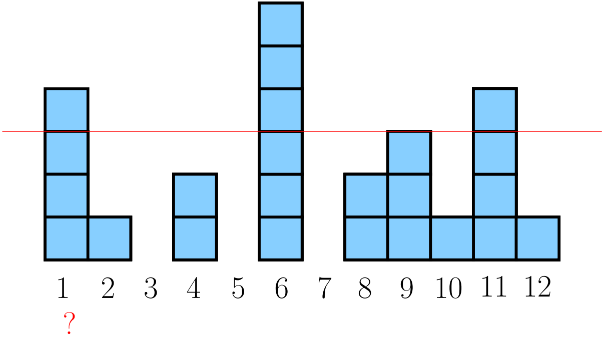
Control event example 2

PSQ Adding a threshold condition (overload condition) on the origin



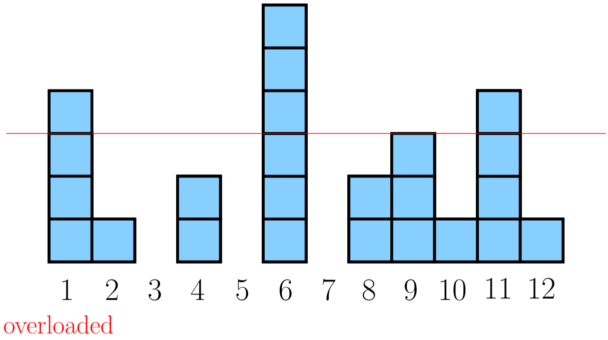
Control event example 2

PSQ Adding a threshold condition (overload condition) on the origin



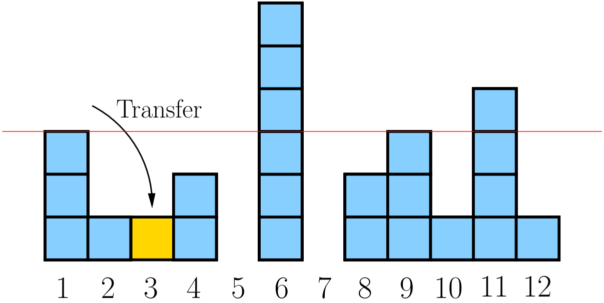
Control event example 2

PSQ Adding a threshold condition (overload condition) on the origin



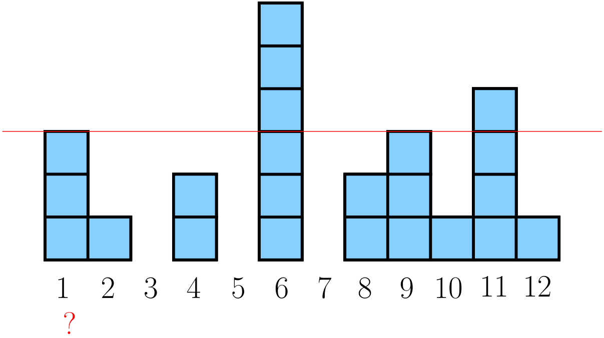
Control event example 2

PSQ Adding a threshold condition (overload condition) on the origin



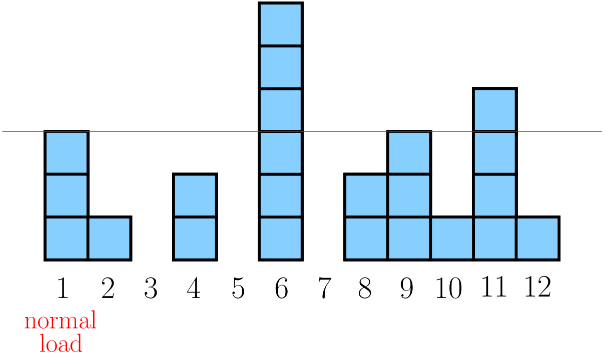
Control event example 2

PSQ Adding a threshold condition (overload condition) on the origin



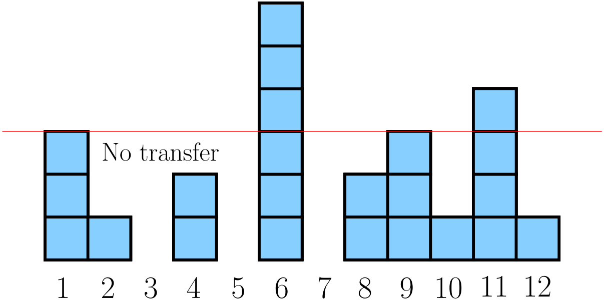
Control event example 2

PSQ Adding a threshold condition (overload condition) on the origin



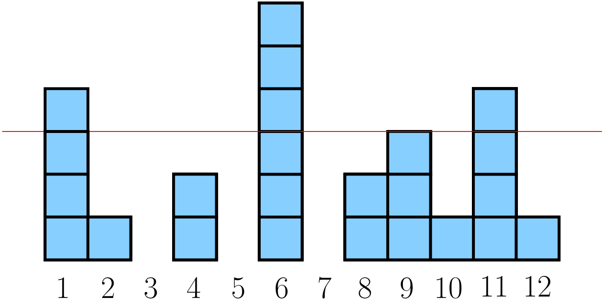
Control event example 2

PSQ Adding a threshold condition (overload condition) on the origin



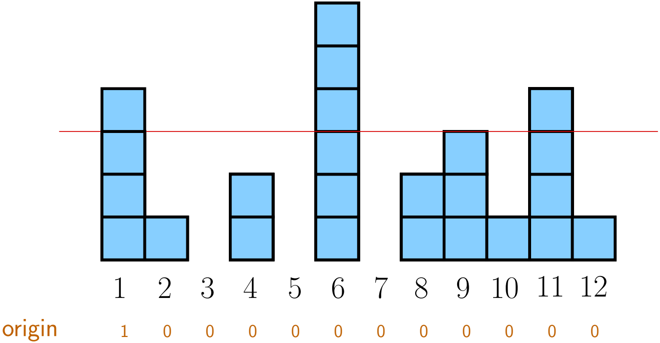
Corresponding index model

PSQ Adding a threshold condition (overload condition) on the origin



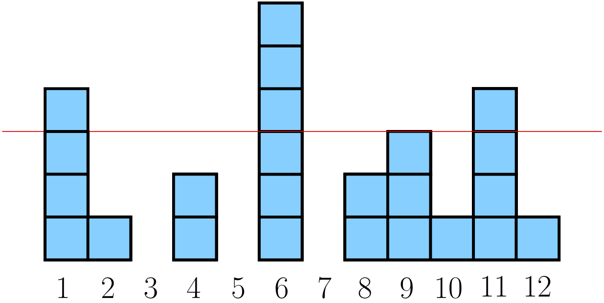
Corresponding index model

PSQ Adding a threshold condition (overload condition) on the origin



Corresponding index model

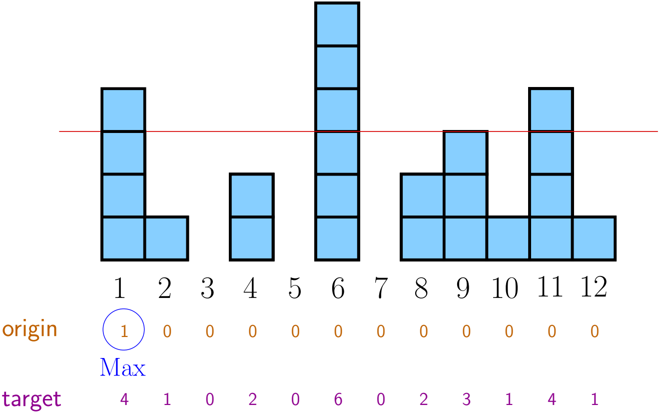
PSQ Adding a threshold condition (overload condition) on the origin



origin	1	0	0	0	0	0	0	0	0	0	0	0
target	4	1	0	2	0	6	0	2	3	1	4	1

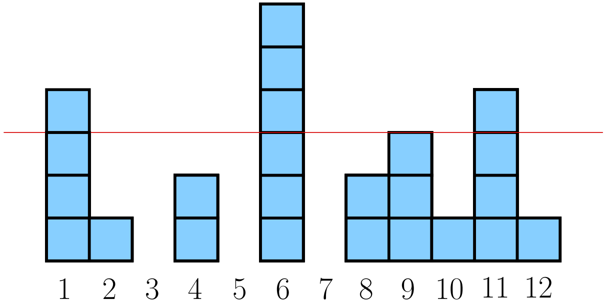
Corresponding index model

PSQ Adding a threshold condition (overload condition) on the origin



Corresponding index model

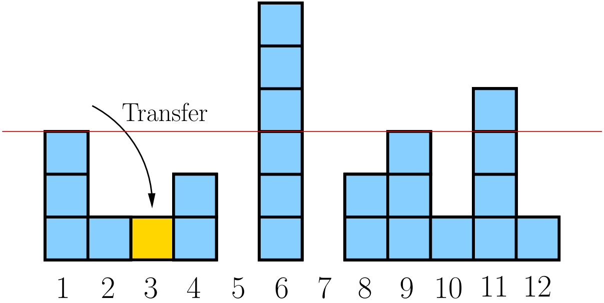
PSQ Adding a threshold condition (overload condition) on the origin



origin	1 (Max)	0	0	0	0	0	0	0	0	0	0	0
target	4	1	0 (min)	2	0	6	0	2	3	1	4	1

Corresponding index model

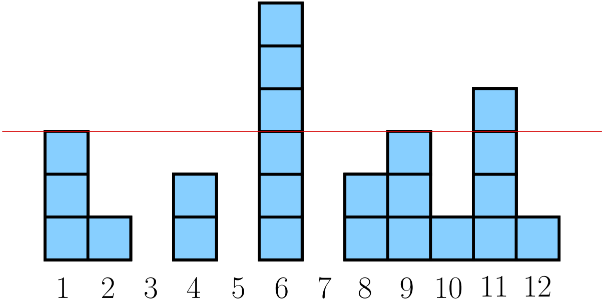
PSQ Adding a threshold condition (overload condition) on the origin



	1	2	3	4	5	6	7	8	9	10	11	12
origin	1 (Max)	0	0	0	0	0	0	0	0	0	0	0
target	4	1	0 (min)	2	0	6	0	2	3	1	4	1

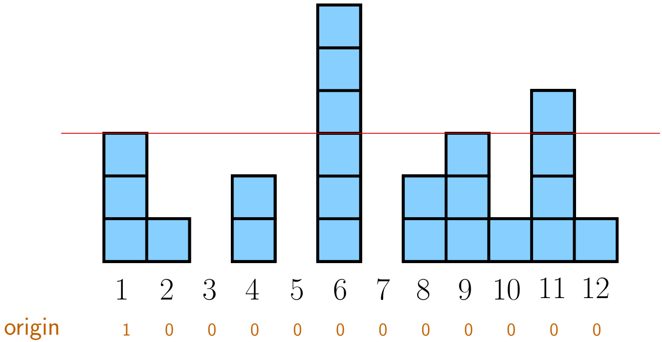
Corresponding index model

PSQ Adding a threshold condition (overload condition) on the origin



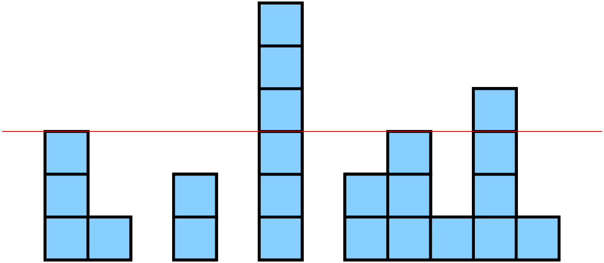
Corresponding index model

PSQ Adding a threshold condition (overload condition) on the origin



Corresponding index model

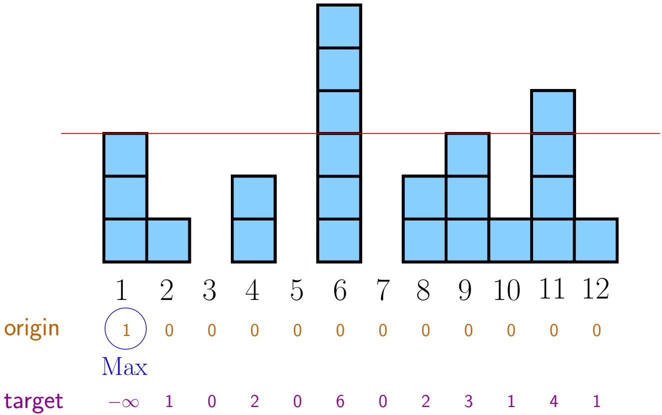
PSQ Adding a threshold condition (overload condition) on the origin



	1	2	3	4	5	6	7	8	9	10	11	12
origin	1	0	0	0	0	0	0	0	0	0	0	0
target	$-\infty$	1	0	2	0	6	0	2	3	1	4	1

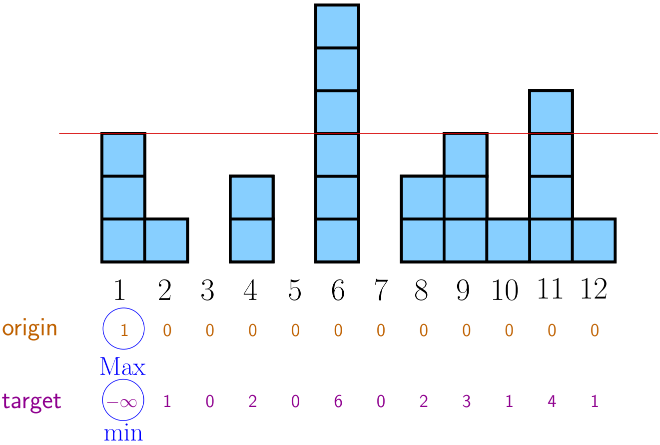
Corresponding index model

PSQ Adding a threshold condition (overload condition) on the origin



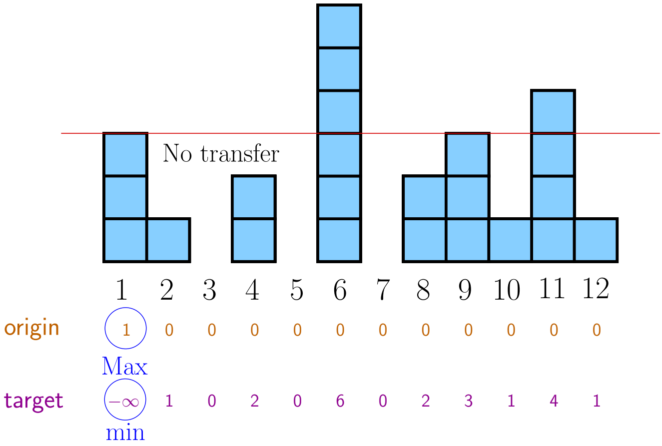
Corresponding index model

PSQ Adding a threshold condition (overload condition) on the origin



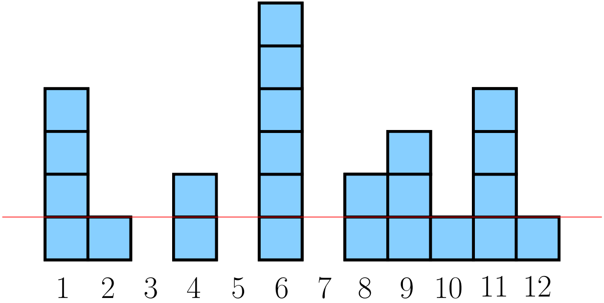
Corresponding index model

PSQ Adding a threshold condition (overload condition) on the origin



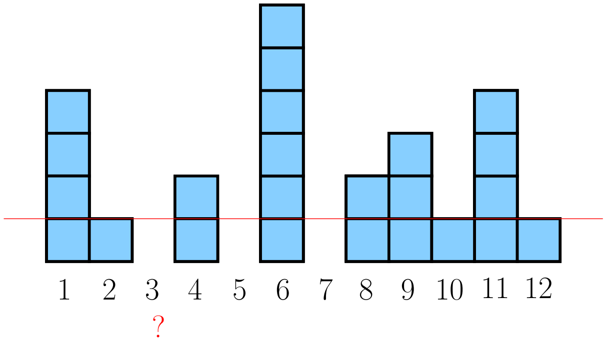
Control event example 3

Pull with probing according to a priority list



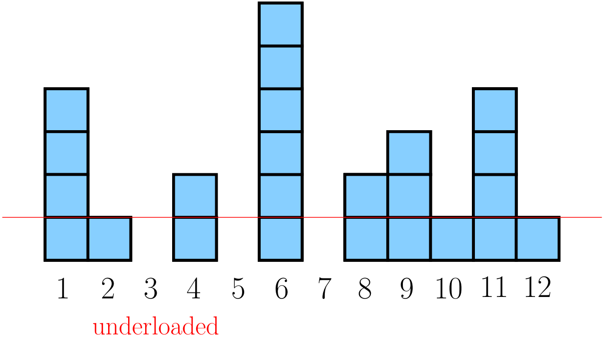
Control event example 3

Pull with probing according to a priority list



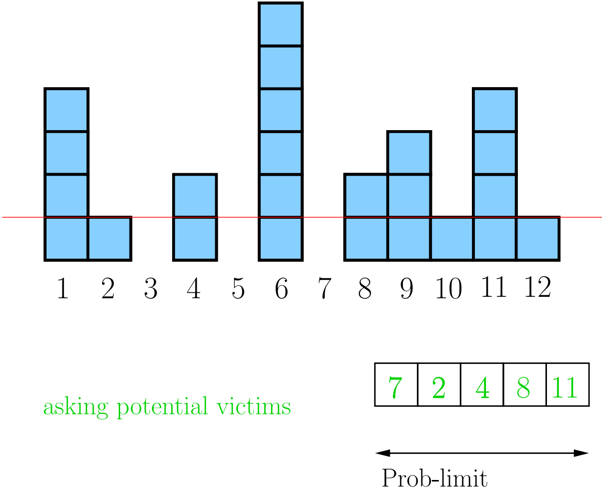
Control event example 3

Pull with probing according to a priority list



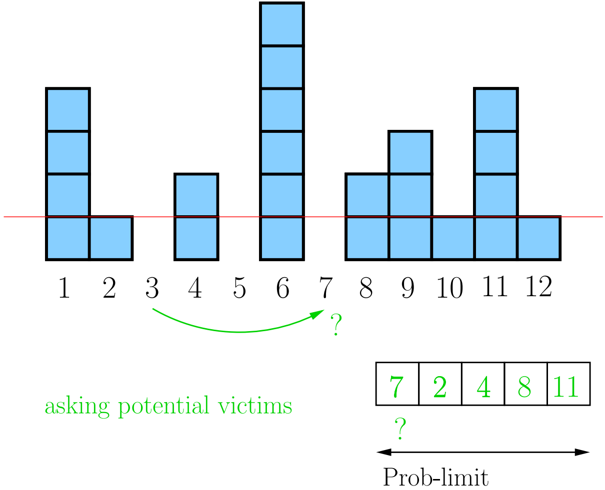
Control event example 3

Pull with probing according to a priority list



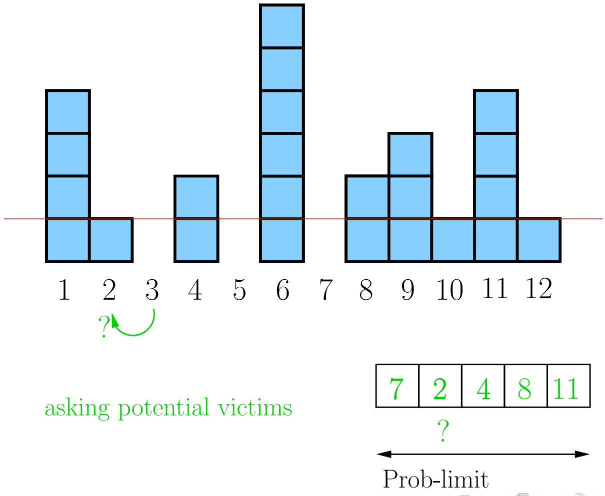
Control event example 3

Pull with probing according to a priority list



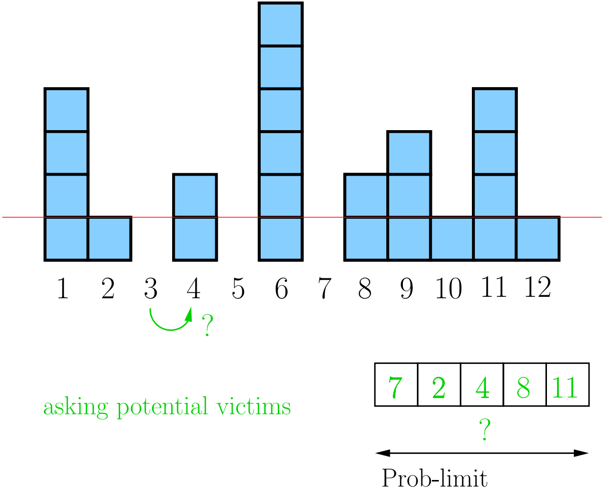
Control event example 3

Pull with probing according to a priority list



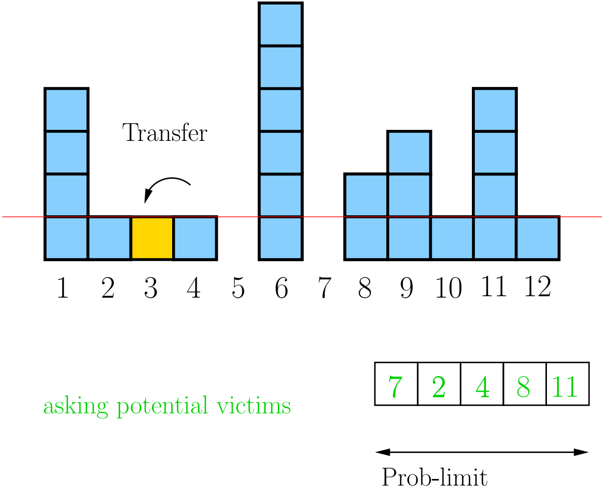
Control event example 3

Pull with probing according to a priority list



Control event example 3

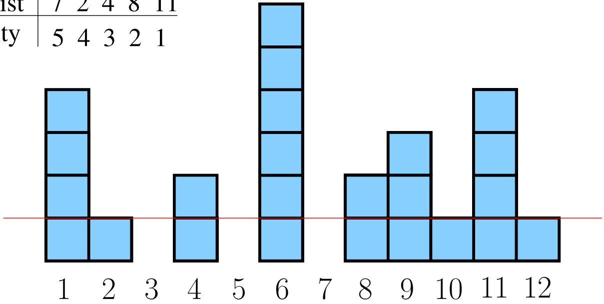
Pull with probing according to a priority list



Corresponding index model

Pull with probing according to a priority list

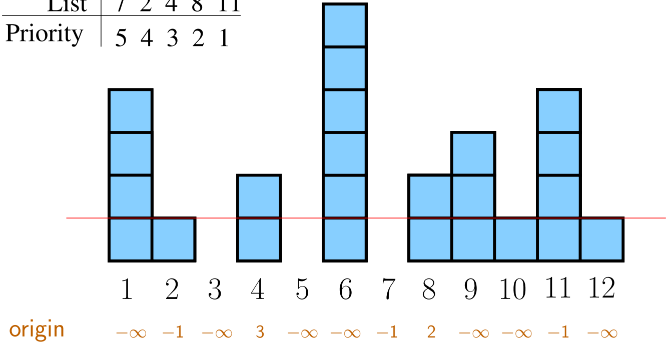
List	7	2	4	8	11
Priority	5	4	3	2	1



Corresponding index model

Pull with probing according to a priority list

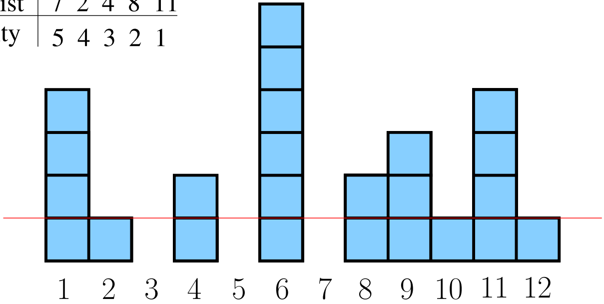
List	7	2	4	8	11
Priority	5	4	3	2	1



Corresponding index model

Pull with probing according to a priority list

List	7	2	4	8	11
Priority	5	4	3	2	1

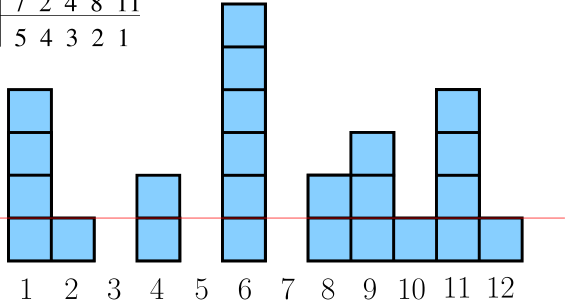


	1	2	3	4	5	6	7	8	9	10	11	12
origin	$-\infty$	-1	$-\infty$	3	$-\infty$	$-\infty$	-1	2	$-\infty$	$-\infty$	-1	$-\infty$
target	0	0	-1	0	0	0	0	0	0	0	0	0

Corresponding index model

Pull with probing according to a priority list

List	7	2	4	8	11
Priority	5	4	3	2	1

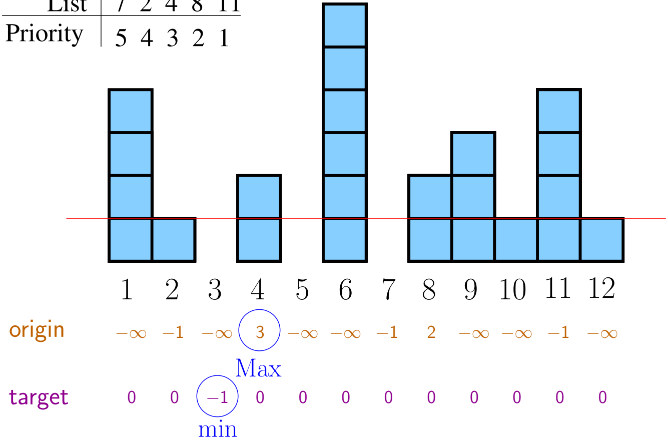


	1	2	3	4	5	6	7	8	9	10	11	12
origin	$-\infty$	-1	$-\infty$	3 (Max)	$-\infty$	$-\infty$	-1	2	$-\infty$	$-\infty$	-1	$-\infty$
target	0	0	-1	0	0	0	0	0	0	0	0	0

Corresponding index model

Pull with probing according to a priority list

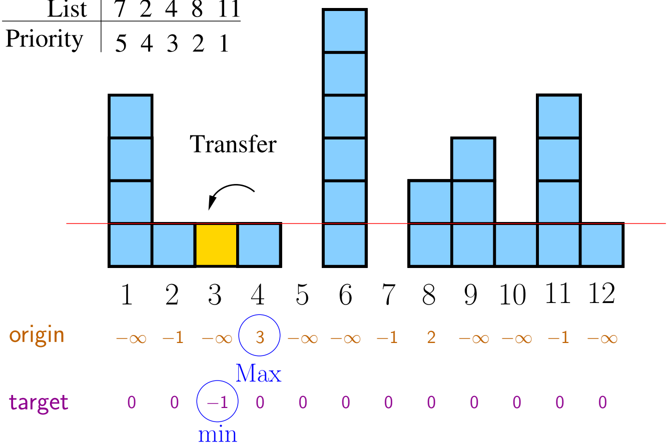
List	7	2	4	8	11
Priority	5	4	3	2	1



Corresponding index model

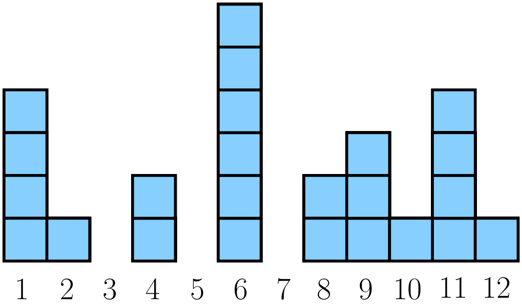
Pull with probing according to a priority list

List	7	2	4	8	11
Priority	5	4	3	2	1



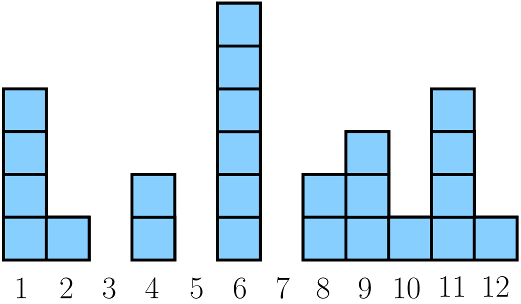
General index model

transfer from an origin (max) to a target (min)



General index model

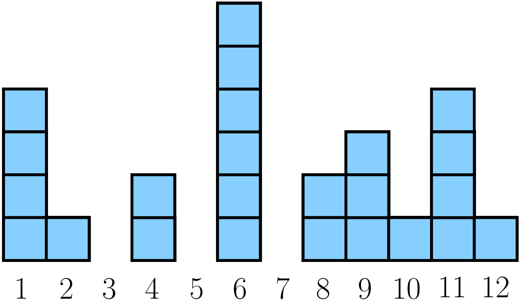
transfer from an origin (max) to a target (min)



origin $I_1(x^1)$ $I_2(x^2)$ $I_3(x^3)$ $I_4(x^4)$ $I_5(x^5)$ $I_6(x^6)$ $I_7(x^7)$ $I_8(x^8)$ $I_9(x^9)$ $I_{10}(x^{10})$ $I_{11}(x^{11})$ $I_{12}(x^{12})$

General index model

transfer from an origin (max) to a target (min)

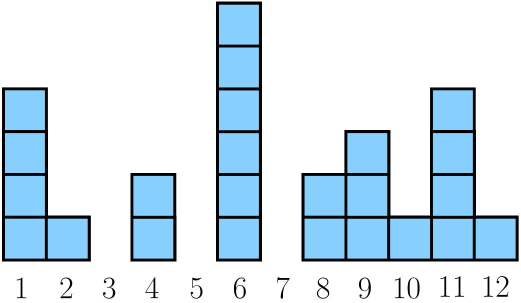


origin $I_1(x^1)$ $I_2(x^2)$ $I_3(x^3)$ $I_4(x^4)$ $I_5(x^5)$ $I_6(x^6)$ $I_7(x^7)$ $I_8(x^8)$ $I_9(x^9)$ $I_{10}(x^{10})$ $I_{11}(x^{11})$ $I_{12}(x^{12})$

target $I_1(x^1)$ $I_2(x^2)$ $I_3(x^3)$ $I_4(x^4)$ $I_5(x^5)$ $I_6(x^6)$ $I_7(x^7)$ $I_8(x^8)$ $I_9(x^9)$ $I_{10}(x^{10})$ $I_{11}(x^{11})$ $I_{12}(x^{12})$

General index model

transfer from an origin (max) to a target (min)



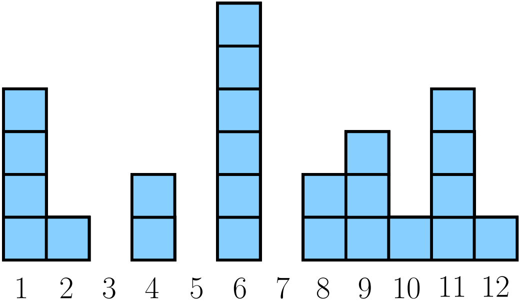
origin $I_1(x^1)$ $I_2(x^2)$ $I_3(x^3)$ $I_4(x^4)$ $I_5(x^5)$ $I_6(x^6)$ $I_7(x^7)$ $I_8(x^8)$ $I_9(x^9)$ $I_{10}(x^{10})$ $I_{11}(x^{11})$ $I_{12}(x^{12})$

Max

target $I_1(x^1)$ $I_2(x^2)$ $I_3(x^3)$ $I_4(x^4)$ $I_5(x^5)$ $I_6(x^6)$ $I_7(x^7)$ $I_8(x^8)$ $I_9(x^9)$ $I_{10}(x^{10})$ $I_{11}(x^{11})$ $I_{12}(x^{12})$

General index model

transfer from an origin (max) to a target (min)



origin $I_1(x^1)$ $I_2(x^2)$ $I_3(x^3)$ $I_4(x^4)$ $I_5(x^5)$ $I_6(x^6)$ $I_7(x^7)$ $I_8(x^8)$ $I_9(x^9)$ $I_{10}(x^{10})$ $I_{11}(x^{11})$ $I_{12}(x^{12})$

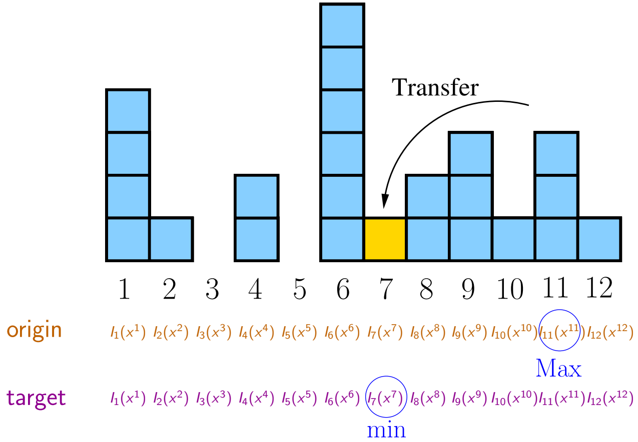
Max

target $I_1(x^1)$ $I_2(x^2)$ $I_3(x^3)$ $I_4(x^4)$ $I_5(x^5)$ $I_6(x^6)$ $I_7(x^7)$ $I_8(x^8)$ $I_9(x^9)$ $I_{10}(x^{10})$ $I_{11}(x^{11})$ $I_{12}(x^{12})$

min

General index model

transfer from an origin (max) to a target (min)



Formalisation

A control event c is defined by :

$$\Phi(x, c) = x - \delta_i + \delta_j$$

i is the **origin**
j is the **target**

Index function

A function $I_k(x^k)$ gives an index, i.e. a cost value to Q_k.

$$i = \mathbf{argmax}_{1 \leqslant k \leqslant K}(I_k^{c,o}(x^k))$$
$$j = \mathbf{argmin}_{1 \leqslant k \leqslant K}(I_k^{c,t}(x^k))$$

Outline

Monotonicity of index load sharing policies

Monotonicity

- $\preceq$ is the natural partial order on the multi-dimensional state space $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_K$.

$$x \preceq y \Leftrightarrow x^i \leqslant y^i \quad \forall i$$

- An event e is monotone if it preserves the partial ordering $\preceq$ on $\mathcal{X}$

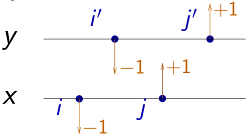
$$\forall (x, y) \in \mathcal{X} \quad x \preceq y \;\Rightarrow\; \Phi(x, e) \preceq \Phi(y, e)$$

Theorem

If all index functions $I_k^{c,o}(x^k)$ and $I_k^{c,t}(x^k)$ are monotone and increasing in function of x^k, then the event c is monotone

Proof

Let $x, y \in \mathcal{X}$ two states with $x \preceq y$,
c a control event, $\Phi(x,c) = x - \delta_i + \delta_j$, $\Phi(y,c) = y - \delta_{i'} + \delta_{j'}$.
Suppose that $i \neq i' \neq j \neq j'$.



Then,

$$I_j^{c,t}(x^j) < I_{j'}^{c,t}(x^{j'})$$ j is the argmin for x

$$I_{j'}^{c,t}(x^{j'}) \leqslant I_{j'}^{c,t}(y^{j'})$$ $I_{j'}^{c,t}$ increasing and $x^{j'} \leqslant y^{j'}$

$$I_{j'}^{c,t}(y^{j'}) < I_j^{c,t}(y^j)$$ j' is the argmin for y

$$I_j^{c,t}(x^j) < I_j^{c,t}(y^j)$$ by transitivity

$$x^j < y^j$$ $I_{j'}^{c,t}$ increasing

$\Rightarrow x^j + 1 \leqslant y^j$, and the order is preserved

Synthesis

Impact of the control triggering :

Triggering policy	Independent control	Application dependent
Push	Monotone	Monotone
Pull	Monotone	Non-monotone

⇒ Almost monotone "Pull on completion" can be simulated with envelopes [BGV08]

Prioritization with index function :

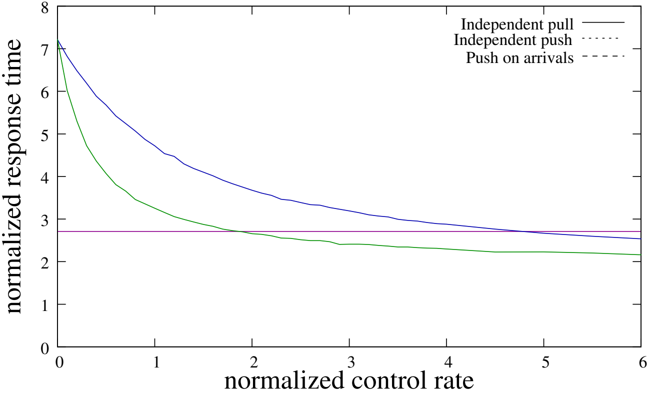
- Using nodes characteristics : CPU speed, capacity ...
- Using system characteristics : network topology
- Threshold criteria
- arbitrary prioritization (priority list)

Random probing : Collection of events with different priority lists

Outline

Estimation of the control rate

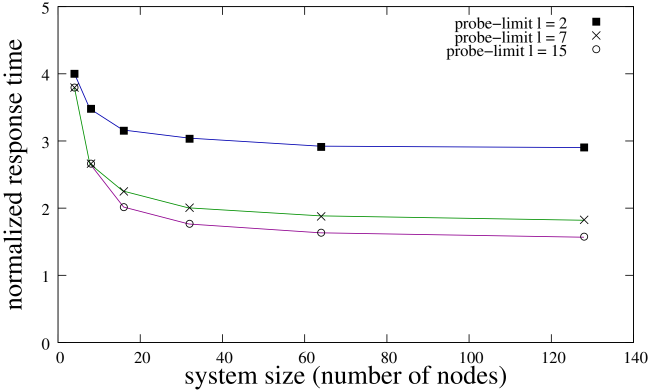
Policy : Controlled Push, Pull and Push on arrivals with random probing of 6 nodes



For a system equipped with a controler, a good operating point is to fix the control rate twice the processor speed.

Estimation of the probe-limit

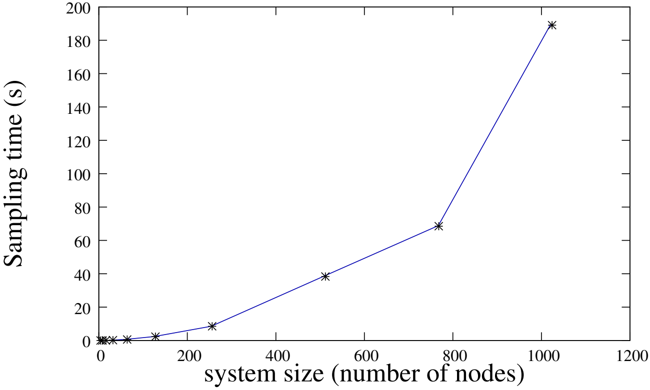
Policy : Controlled Push with random probing of 6 nodes



increasing the Probe-limit further than 7 does not provide a significant performance improvement

Scaling

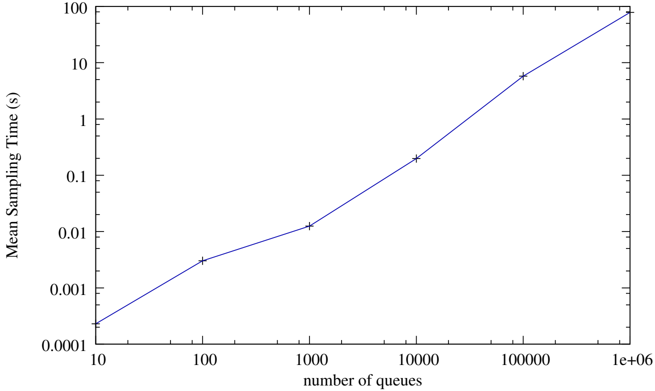
Policy : Controlled Push with random probing of 6 nodes



It is feasible to simulate complex load sharing strategies within a system of 1024 nodes.

Scaling Toward million of nodes

Policy : Threshold Push on Arrival with priority list of 8 nodes



The time to simulate such system is linear with the number of nodes

Outline

Conclusion

Monotonicity result

A framework for modelling monotone control events in load sharing systems

Efficient performance evaluation method and tool

Advantages

- unbiased sampling
- large scale models

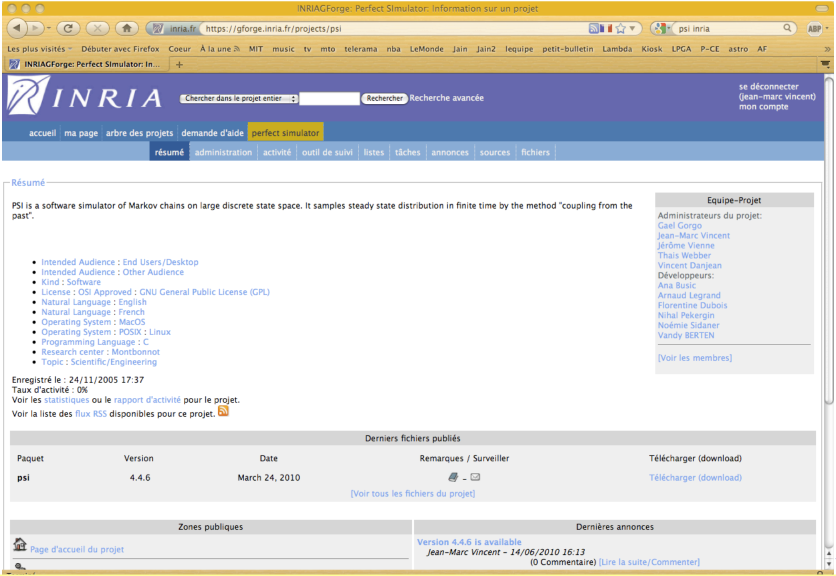
Limitations

- Markovian assumptions everywhere
- No synchronization between jobs
- Migration time neglected

Future work

- Almost monotone events
  $\Rightarrow$ Pull on completion is an almost monotone event and can be simulated with Envelope techniques [BGV08]
- Model refinements
  $\Rightarrow$ Considering transfer costs, more realistic distributions (arrivals, completions)
- More experiments with large scale systems
  $\Rightarrow$ Understanding the behaviour of large scale systems

Download : http ://gforge.inria.fr/projects/psi



References I

A. Bušić, B. Gaujal, and J.M. Vincent, *Perfect simulation and non-monotone markovian systems*, ValueTools '08 : Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools, 2008, pp. 1–10.

M. Béguin, L. Gray, and B. Ycart, *The load transfer model*, The Annals of Applied Probability **8** (1998), no. 2, 337–353.

P. Bremaud, *Markov chains, gibbs fields, monte carlo simulation and queues*, Springer, 1999.

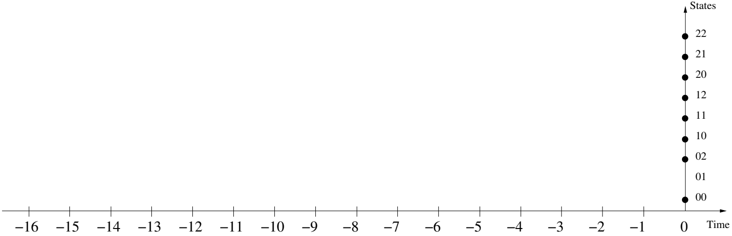
S.P. Dandamudi, M. Kwok, and C. Lo, *A comparative study of adaptive and hierarchical load sharing policies for distributed systems*, Computers and Their Applications, 1998, pp. 136–141.

D.L. Eager, E.D. Lazowska, and J. Zahorjan, *A comparison of receiver-initiated and sender-initiated adaptive load sharing*, Performance Evaluation **6** (1986), no. 1, 53–68.

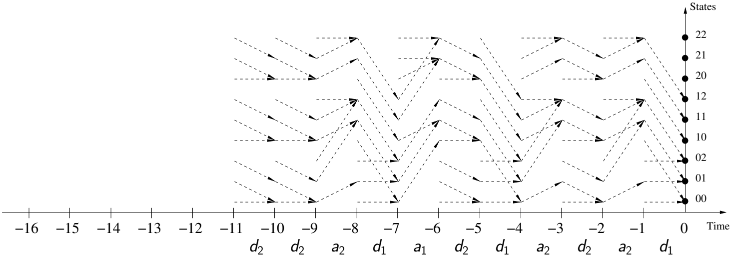
References II

- H. D. Karatza and R. C. Hilzer, *Parallel and distributed systems : load sharing in heterogeneous distributed systems*, Winter Simulation Conference, 2002, pp. 489–496.
- M. Mitzenmacher, *Analyses of load stealing models based on differential equations*, Symposium on Parallel Algorithms and Architectures, 1998, pp. 212–221.
- R. Mirchandaney, D. Towsley, and J.A. Stankovic, *Adaptive load sharing in heterogeneous distributed systems*, Journal of Parallel and Distributed Computating **9** (1990), no. 4, 331–346.
- J.G. Propp and D.B. Wilson, *Exact sampling with coupled markov chains and applications to statistical mechanics*, Random Structures and Algorithms **9** (1996), no. 1-2, 223–252.

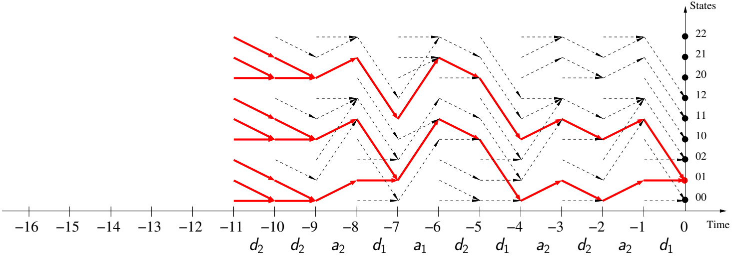
Perfect sampling algorithm



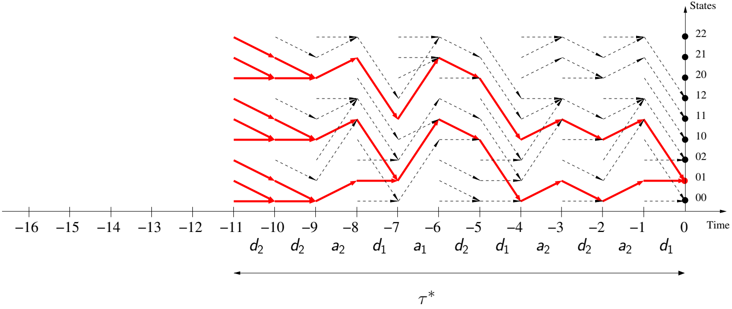
Perfect sampling algorithm



Perfect sampling algorithm



Perfect sampling algorithm

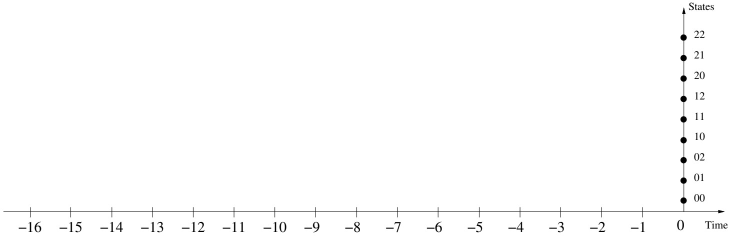


Average time complexity

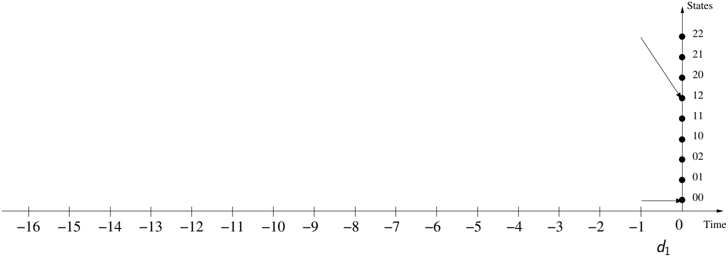
C_Φ mean computation cost of $\Phi(x, e)$ $C = Card(\mathcal{X}).\mathbb{E}(\tau^*).\rfloor_\oplus$

On the example : $C = 9 * 11 = 99$

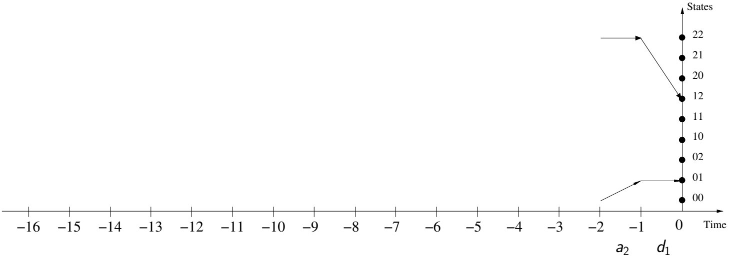
Monotone perfect sampling algorithm



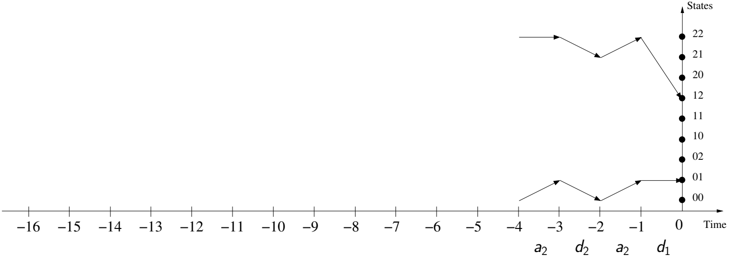
Monotone perfect sampling algorithm



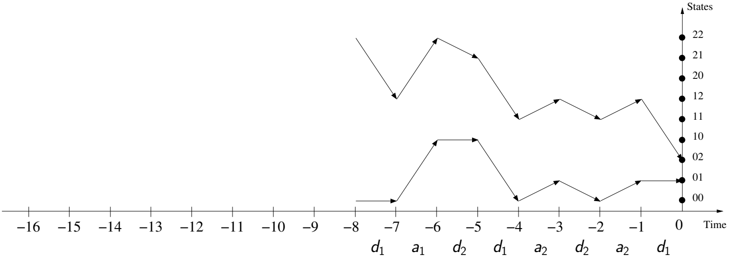
Monotone perfect sampling algorithm



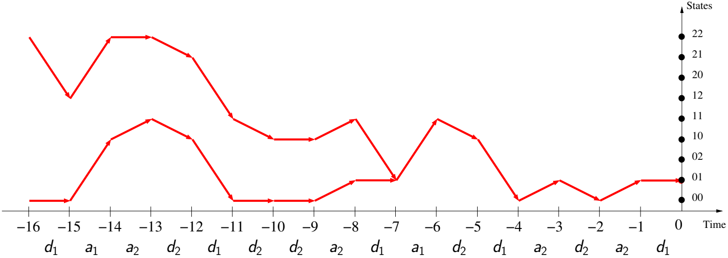
Monotone perfect sampling algorithm



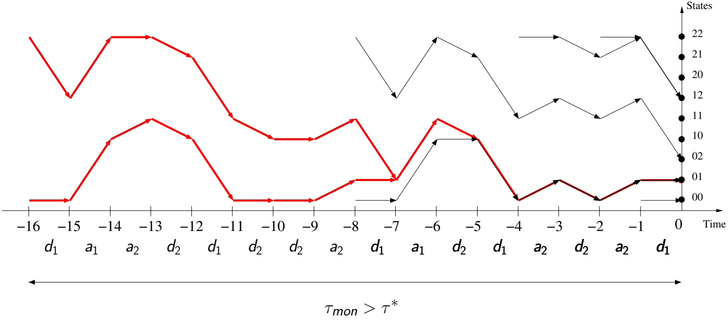
Monotone perfect sampling algorithm



Monotone perfect sampling algorithm



Monotone perfect sampling algorithm



Average time complexity

$C_{mon} \leqslant 2.(2.\mathbb{E}(\tau_{mon}))$

$\Rightarrow$ Reduction factor of $\frac{4}{Card(\mathcal{X})}$

On the example : $C_{mon} = 2 * (1 + 2 + 4 + 8 + 16) = 62$