PDP09

Byte-Range Asynchronous Locking in Distributed Settings

Martin Quinson
UHP-Nancy I
LORIA

&

Flavien Vernier
Univ. Savoie
LISTIC

Context & Objectives

- Context
  - Mutual exclusion algorithms
  - Large scale distributed systems
  - Large resource
- Objectives
  - Request partialy a critical section
  - Asynchronous range locks

Contents

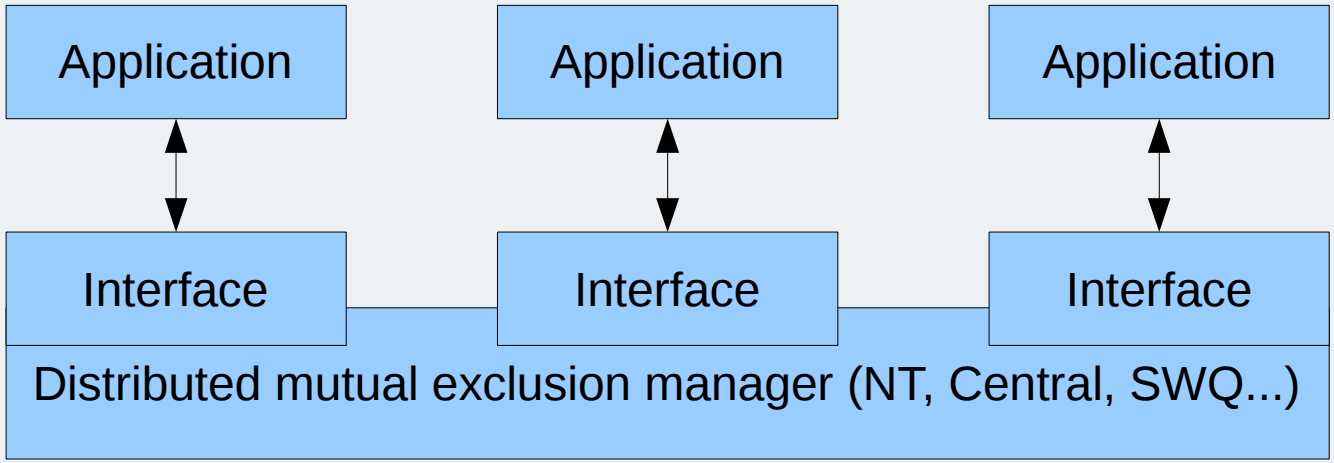
- Asynchronous Range Locking
- Naimi-Trehel Algorithm
- Split Waiting Queues (SWQ) algorithm
  - The algorithm
  - A use case
- Experiments
  - Comparison between: classical Naimi-Trehel, centralized and SWQ algorithms
- Conclusion & Future works

Asynchronous Ranged Locking

- Asynchronous Range Locking is the ability to:
  - Partially lock a resource,
  - not be blocked by the request of a lock.
- It is very useful:
  - if the access to the resource can be predicted,
  - to continue to work until access to the resource.
- This approach is particularly adapted to distributed systems and large resource.

Asynchronous Ranged Locking

- Interface:
  - lockCreate: range → lockId
  - lockTest: lockId → bool
  - doLock: lockId → data
  - unLock: lockId x data → null



Naimi-Trehel Algorithm

- Naimi-Trehel algorithm can be split in two parts:
  - The token request
    - owner (peer): the 'supposed' owner
    - request (bool): true if the node requests the token.
  - The token exchange
    - token (bool): true if the node has the token.
    - next (peer): the peer to which the token must be sent.
- These two parts are strongly linked

Split Waiting Queues (SWQ) algorithm

- SWQ algorithm uses the same principles as Naimi-Trehel algorithm.
- SWQ algorithm introduces a new concept in Naimi-Trehel algorithm:
  - The token can be split in sub-tokens.
  - The sub-tokens are tokens.
  - Two contiguous tokens can be merged in only one token.
- A token is a contiguous range.

Split Waiting Queues (SWQ) algorithm

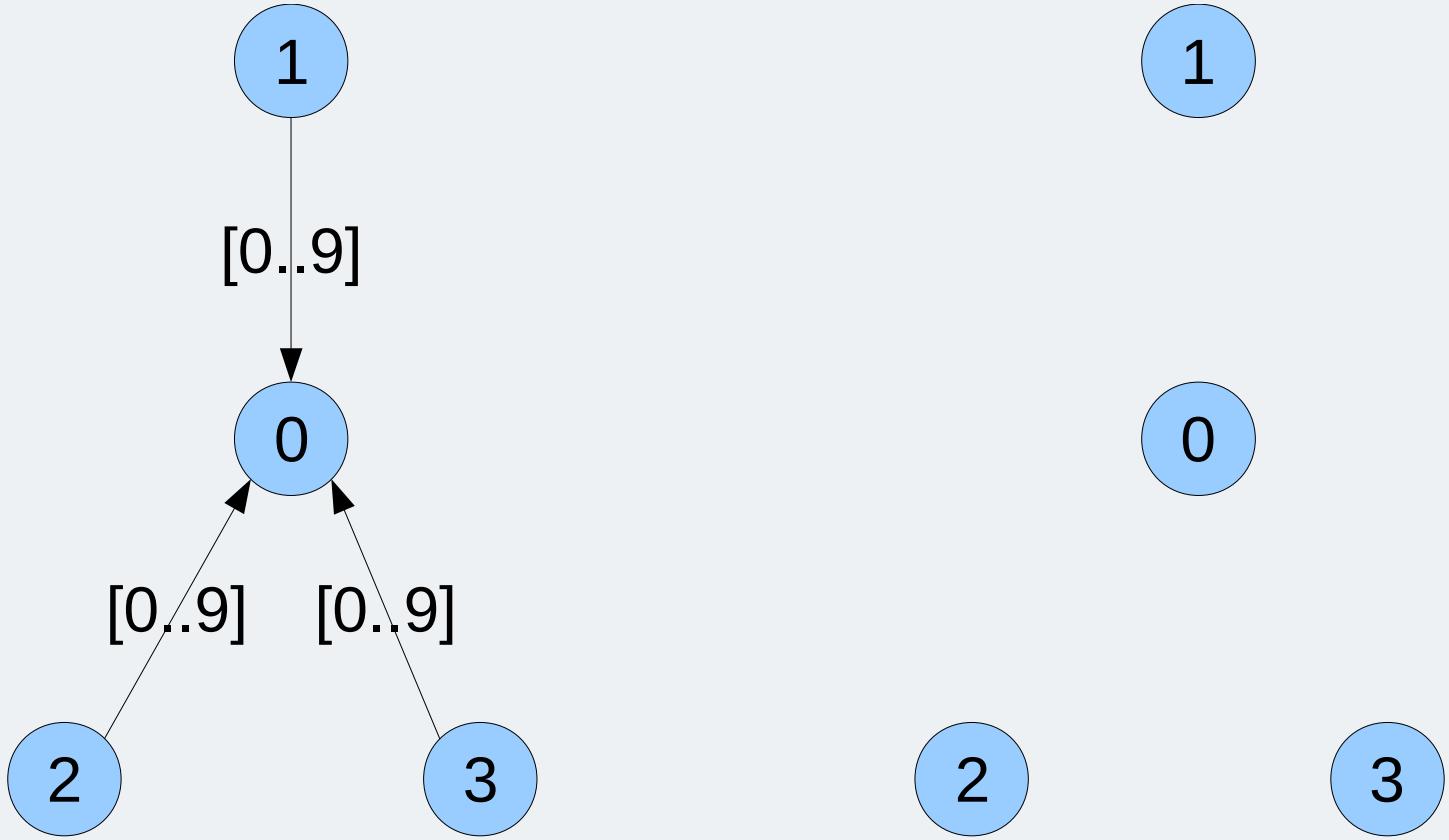
- To manage the multi-tokens the information stored by Naimi-Trehel algorithm are modified:
  - owners (list of peers): the list of 'supposed' owners
  - owned (list of tokens): the stored tokens
  - request (range): the requested token.
  - nexts (list of peer-token): the peers to which the tokens should be sent.

Split Waiting Queues (SWQ) algorithm

- To avoid conflicts or dead locks:
  - two supplementary lists must be managed:
    - requestingReceived (list of tokens): sub-tokens of the request already received
    - waitingRequests (list of requests): the requests that conflict with its own request.
  - a request is always proceeded sequentially,
  - a request message contains the requested range and the already found range,
  - A node becomes the owner of a part only when its request finds this part.

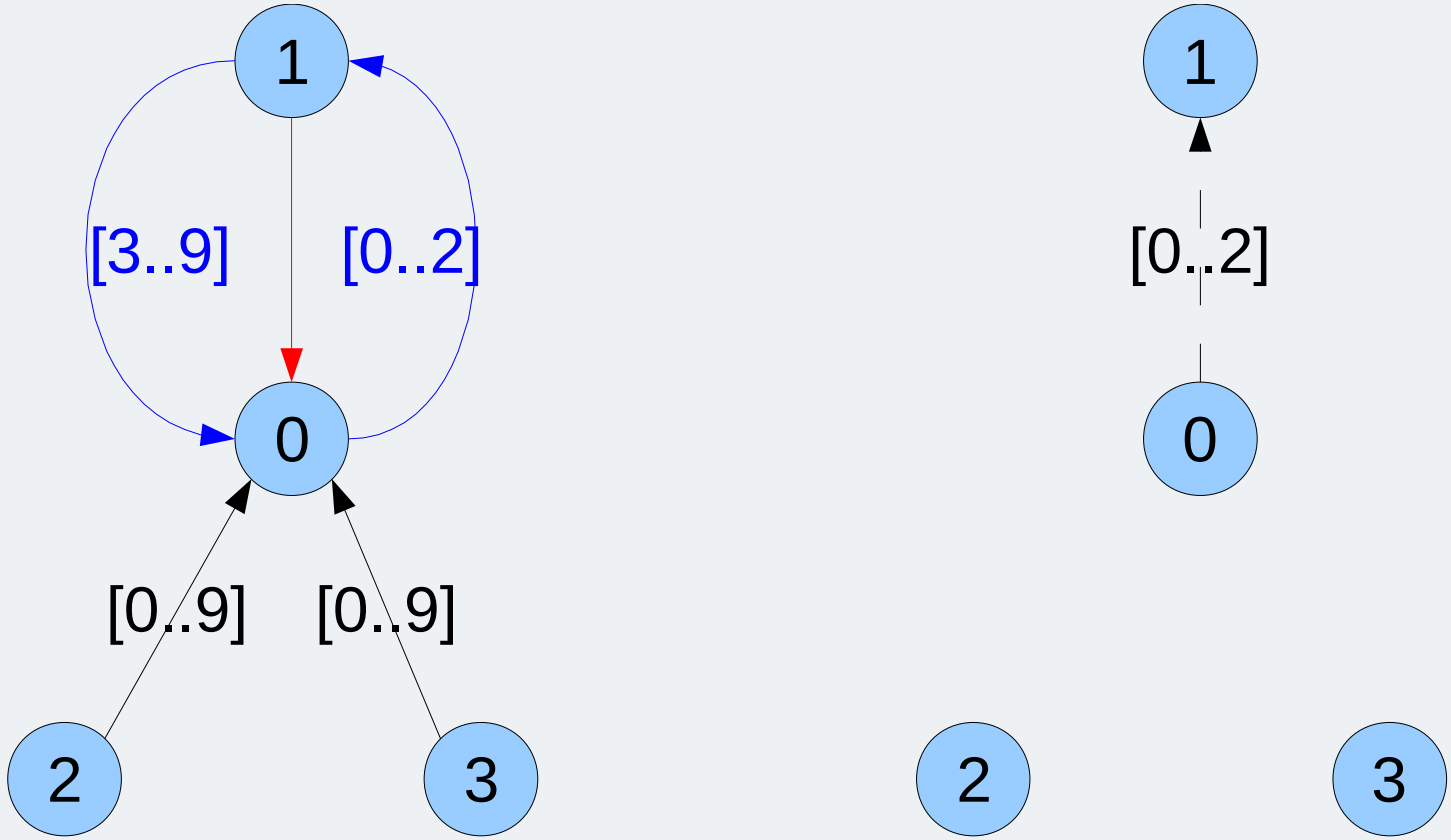
Split Waiting Queues (SWQ) algorithm: illustration

- Initial configuration:



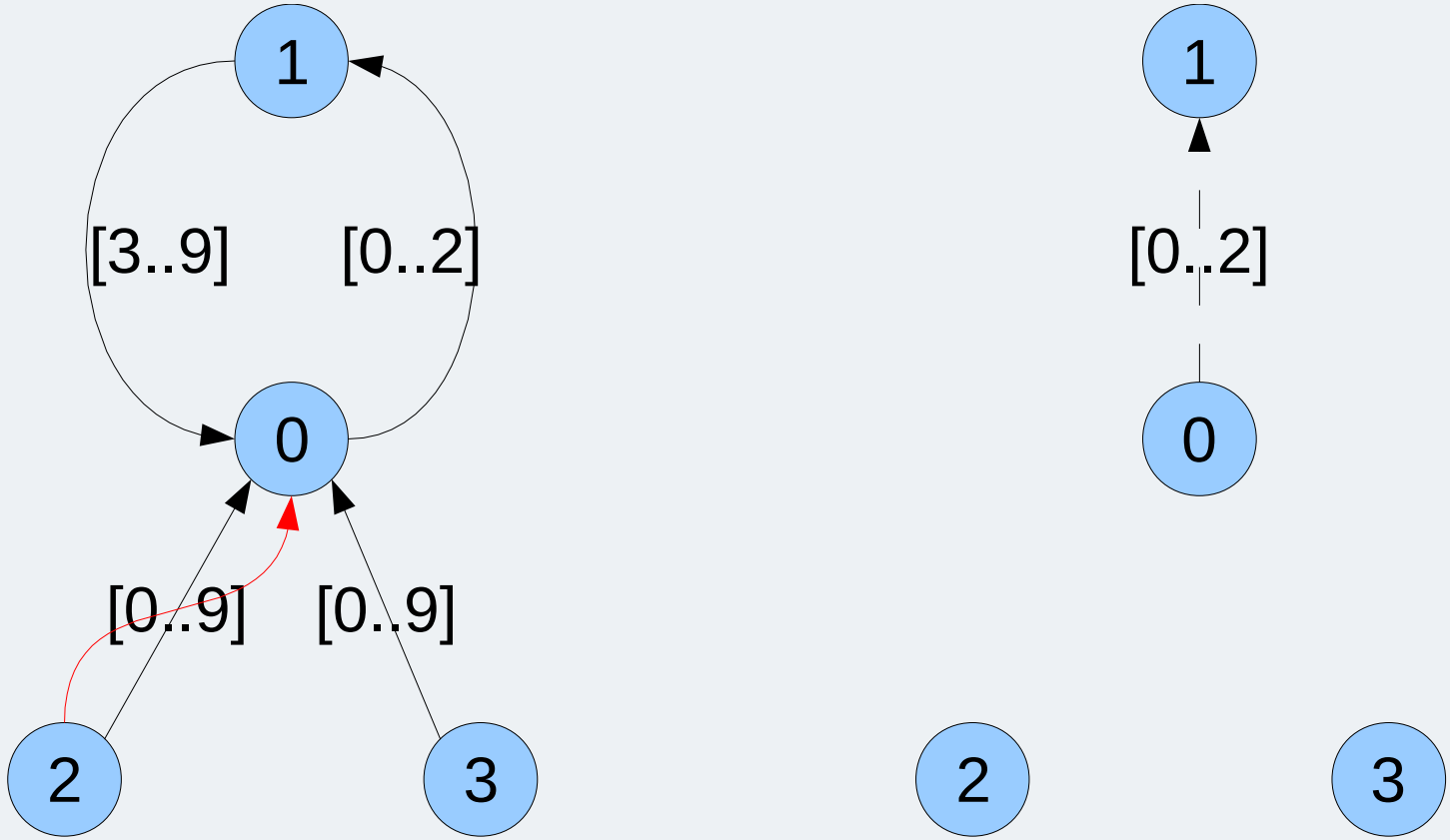
Split Waiting Queues (SWQ) algorithm: illustration

- Node 1 requests [0..2]:



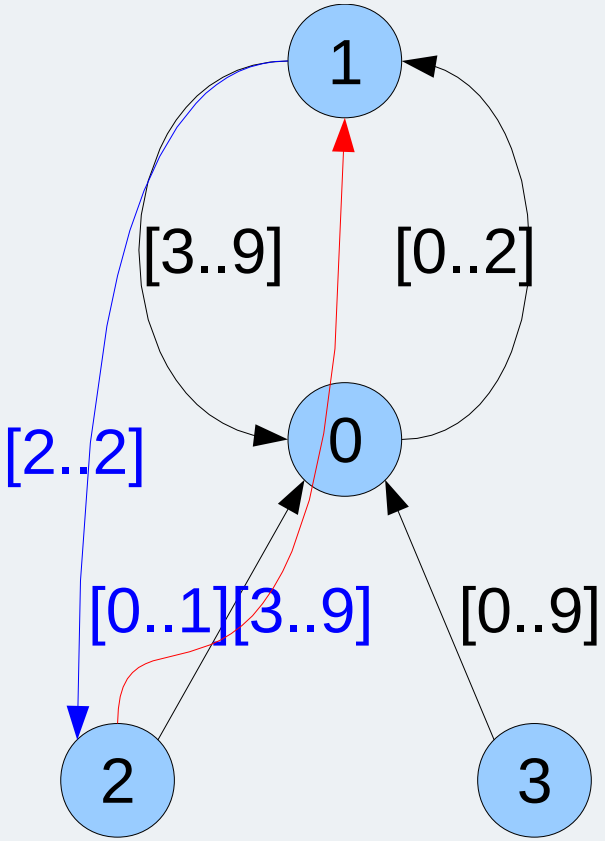
Split Waiting Queues (SWQ) algorithm: illustration

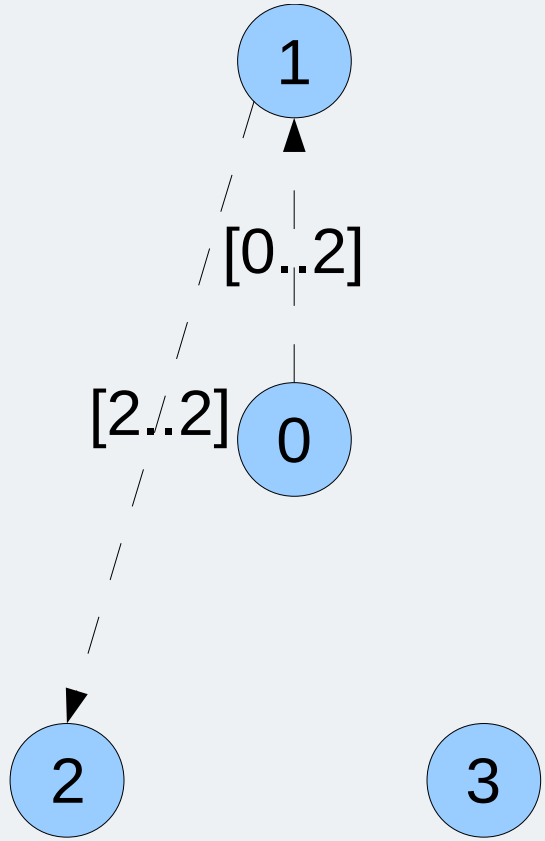
- Node 2 requests [2..3]:



Split Waiting Queues (SWQ) algorithm: illustration

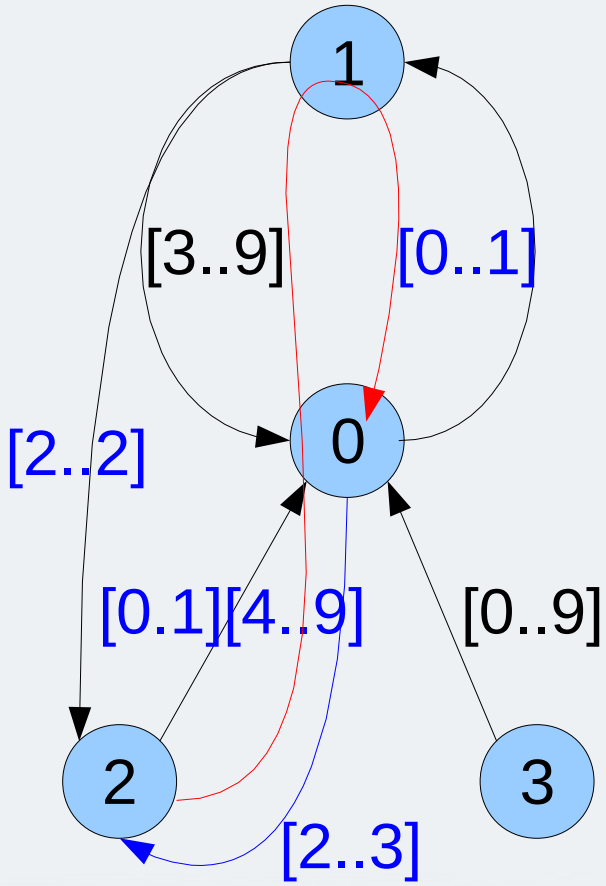
- Node 2 requests [2..3]:

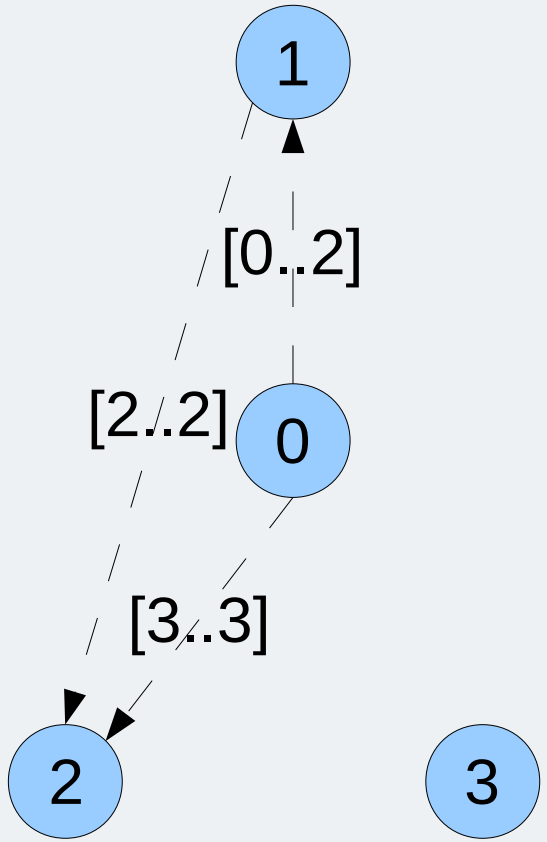




Split Waiting Queues (SWQ) algorithm: illustration

- Node 2 requests [2..3]:





Experimentations

- The experiments are realized with the GRAS API of SimGrid Simulator.
- Each node requests 25 locks.
- The parameters used are:
  - alpha: the time each lock is kept by nodes,
  - the number of nodes,
  - the size of the resource,
  - the size of requested range.

Experimentations: first comparison

- The resource size is fixed to 64kb.

alpha	range	algo	# of nodes					
			2	12	22	42	82	162
0.25	1/16	central	0.2	1.6	3.6	7.6	16.3	33.5
		SWQ	0.3	3	5	11	32.6	117
	full	NT	0.8	13.1	23.4	44.7	82.6	165
		central	1	19.1	37.8	70.2	138	277
		SWQ	0.9	26.6	72.1	228	745	2807
5	1/16	central	0.6	5.1	9.7	18.9	37.2	73.1
		SWQ	0.6	6.2	13.1	31.3	60.9	153
	full	NT	4.5	64.8	122	237	463	922
		central	5.7	71.7	137	264	520	1035
		SWQ	4.5	64.8	122	268	787	2860

Experimentations: Impact of resource size

- Range fixed to 1/16

size	alpha	algo	# of nodes					
			2	12	22	42	82	162
640kb	0.25	central	0,4	5,3	11,5	21,8	43,9	86,4
		SWQ	0,5	4,3	6,4	14,2	36	124
	5	central	0,7	7	13,4	24,4	47,7	92,9
		SWQ	0,7	7,3	14	32,3	66,4	151
6.25Mb	0.25	central	4.9	48.2	92.8	165	318	618
		SWQ	2.6	21	27.1	50.8	91.8	199
	5	central	4.9	46	88.7	159	312	618
		SWQ	2.6	19.9	30.3	66	113	240

Conclusion & Future works

- A new distributed mutual exclusion algorithm adapted to distributed system and large resource is introduced:
  - It allows partial locks,
  - It has not central point,
  - It allows asynchronous lock.
- The main future works will consist in:
  - comparing the simulation results to live deployements,
  - relaxing some constraints to improve the performances