Performance stabilization of a token based epidemic diffusion

Augusto Ciuffoletti
INFN/CNAF - Italy

WRAS 2007 - Paris

The problem and our solution

PROBLEM

- Maintain shared knowledge
- Requirements:
  - predictable (low) overhead
  - predictable update latency
  - expandability
  - scalability

SOLUTION

- Use an epidemic diffusion pattern
- Diffusion is supported by “wandering” tokens
- The number of tokens adapts to system size

Self stabilization issues

- The number of tokens stabilizes around a value which depends on:
  - the size of the system (variable)
  - the required update latency (constant)

Token Number = Number of units / Latency

- Token loss events are managed with the same mechanism used to introduce new tokens when system grows
- Presence of spurious tokens is managed with the same mechanism used to remove tokens when system shrinks

Extended use of probabilistic techniques

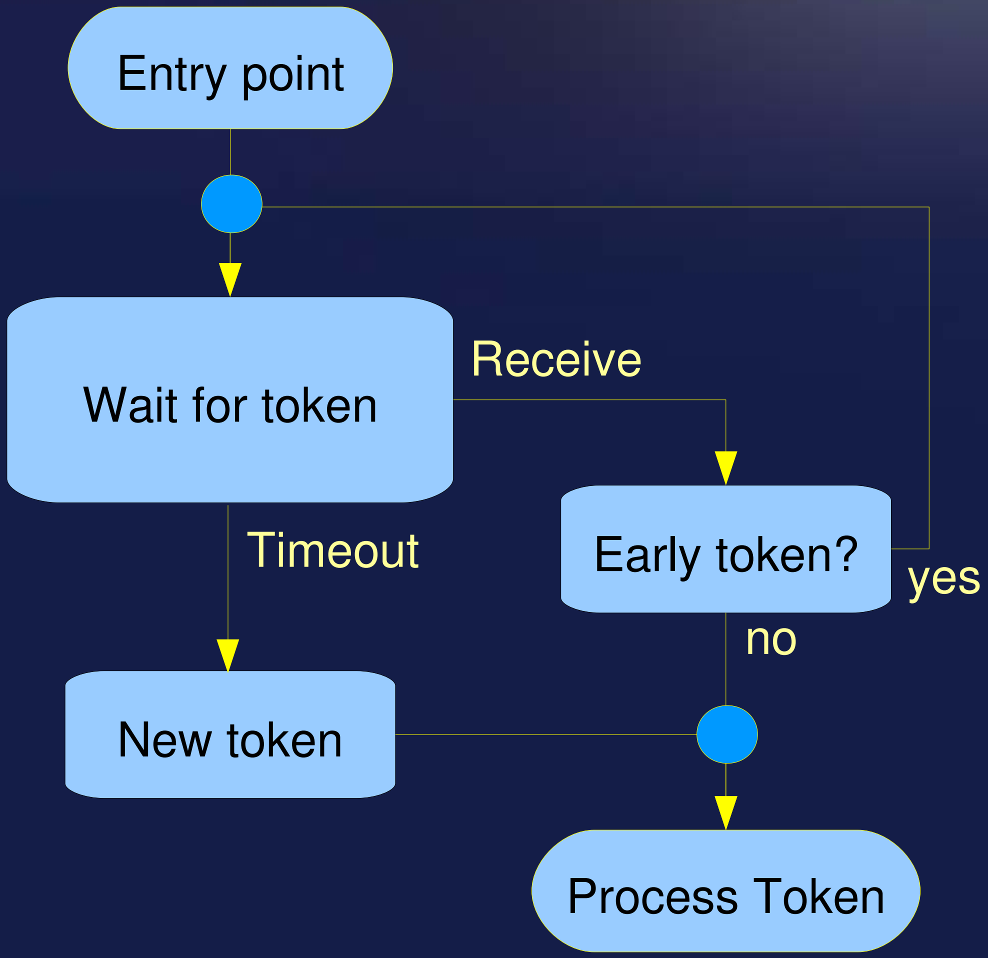
Probabilistic technique

=

success with high probability

- Wandering token ensures a fair behavior with high probability
- Timing ruled token generation ensures a stable regulation of the number of tokens with high probability
- Timing ruled token removal ensures a stable regulation of the number of tokens with high probability

Regulate number of tokens in the system



Expected token interarrival:

T_i=Latency*O(1-p_{fail})

(extended formula in paper)

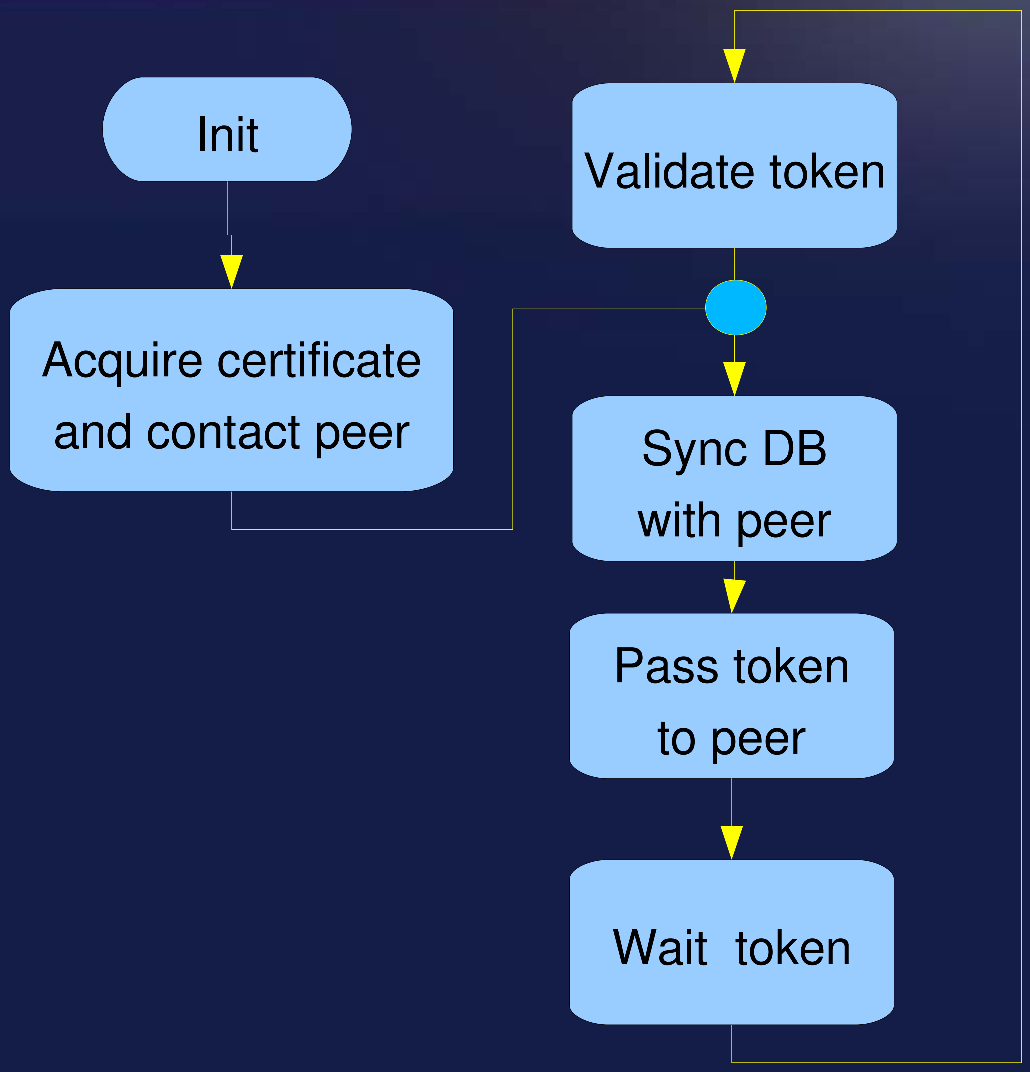
Timeout = T_i*3

Early token threshold: $T_i/3$

NOTE:

Flow control depends only on required latency and reliability

Self-referential use case: membership



- An Internet transport level token circulation needs the availability of a registry of all members
- One way to propagate changes to the membership is syncing the databases of the peers exchanging the token

Complexity of synchronization

- Database synchronization is needed to:
  - propagate join/leave events
  - maintain a database of public keys
- In order to limit its footprint, we consider that (except for initialization) each synchronization operation requires the transfer of a limited number of events (the "capacity" of the token)
- Each host regulates the number of new events included in a synchronization operation (FIFO)
- The frequency of updates on a single host is limited:

Inter-arrival=(Latency*Size)/Capacity

- The inter-arrival time depends on system size!

A scalable rule to regulate update frequency

- The number of updates during one single synchronization is limited by a system wide value
- A stack of updates (of limited size) is maintained, and governed FIFO
- A new push occurs at times that are determined using a randomized rule
- For each token visiting the host, a pending update is injected with probability:

(Latency*Tokens)/(Capacity*TokenLatency)

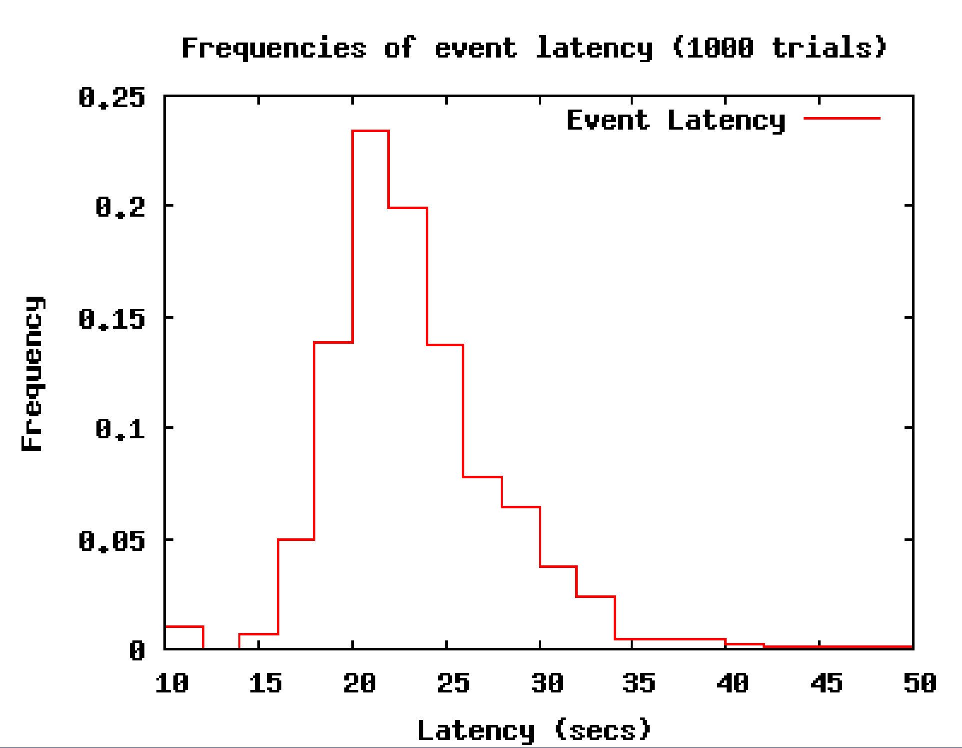
- Such rule probabilistically ensures that each update has enough time to reach every host

Intrinsically randomized

- Randomized decision:
  - to compensate token loss
  - to compensate token duplication
  - to stabilize event latency
  - to stabilize system load
- An analytic verification is awkward: we opt for a simulation
- Parameters:

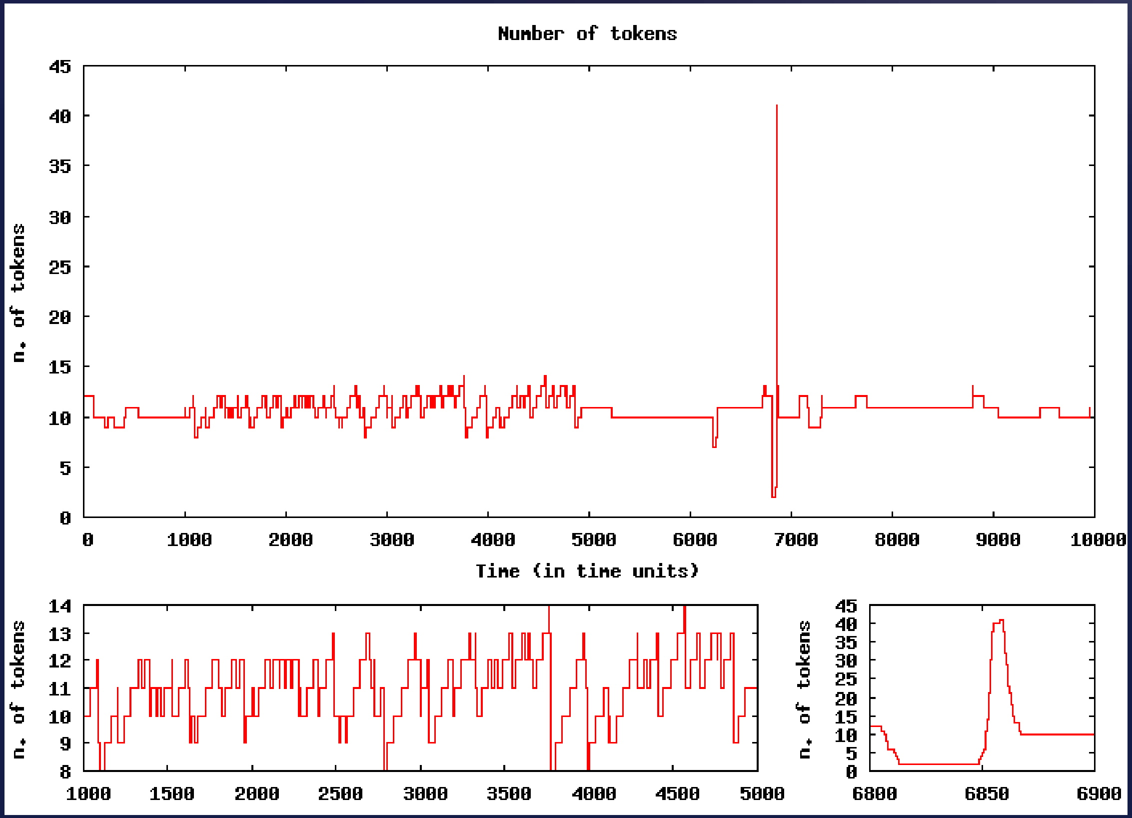
Latency	40 secs
P_{fail}	0.1%
Token latency	30 msec
Capacity	10 events

Event Latency without self regulation



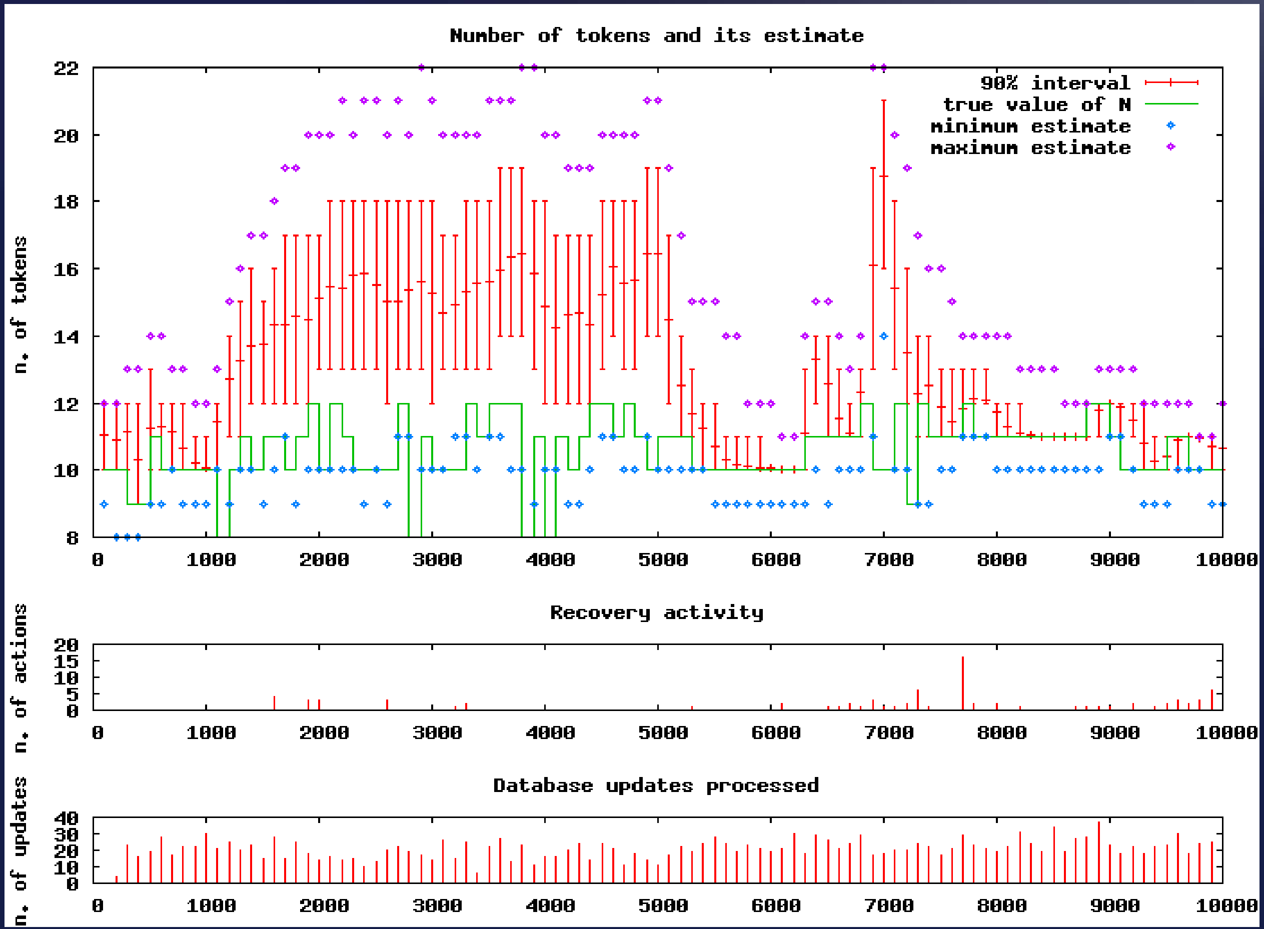
- Event Latencies in a system of 1000 units containing the expected number of token (indeed, 11 instead of 11.4)
- No regulation: we just check whether the Event Latency falls above 40 secs a number of times compatible with a probability of 0.1%
- The frequency is in fact slightly above that: about 0.5%

Token Number Regulation



- Duration 10000 seconds (2h 45')
- We need to keep the number of tokens around 11
- We inject a massive join from time 1000 to time 5000 to test stability (100 units join)
- An unexpected event around time 6800

Local estimate of the number of tokens



- This value is used to regulate update frequency.
- During normal operation, hosts perceive a number of tokens slighly higher than real
- During growth transient that gap increases
- The number of updates per period is quite stable

Conclusions

- We have explored the potential of a technique useful to maintain a shared database
- The membership sharing such knowledge is variable, one application of our algorithm is its maintenance
- We made extensive use of probabilistic rules: the result is an algorithm with a extremely low overhead, and characterized by an extreme scalability
- Its behavior cannot be analyzed formally above the first order characteristics: we propose a series of simulations
- The results prove that the behavior of the system is quite adherent to expectations (first order) and stable
- More investigation needed...