Ordem Total Optimista em Redes WAN

Antônio Sousa, José Pereira,
Francisco Moura, Rui Oliveira

Sumário

- Objectivo e visão geral do protocolo
- Ordem total
- Ordem total optimista
- Obstáculos para ordem total espontânea
- Protocolo proposto
  - Funcionamento
  - Teste de performance
  - Validação dos resultados
  - Conclusões

Objectivo

- Propor um protocolo que assegure que todos os nós receberão as mensagens optimistas ao mesmo tempo.

Visão geral do protocolo

- Utiliza algoritmo de ordem total baseado em seqüenciador
- Introduz delay para que todos os nós recebam as mensagens optimistas ao mesmo tempo e, conseqüentemente, na mesma ordem que as autoritárias
- Cada mensagem é entregue duas vezes para cada nó: optimista e autoritária

Ordem total

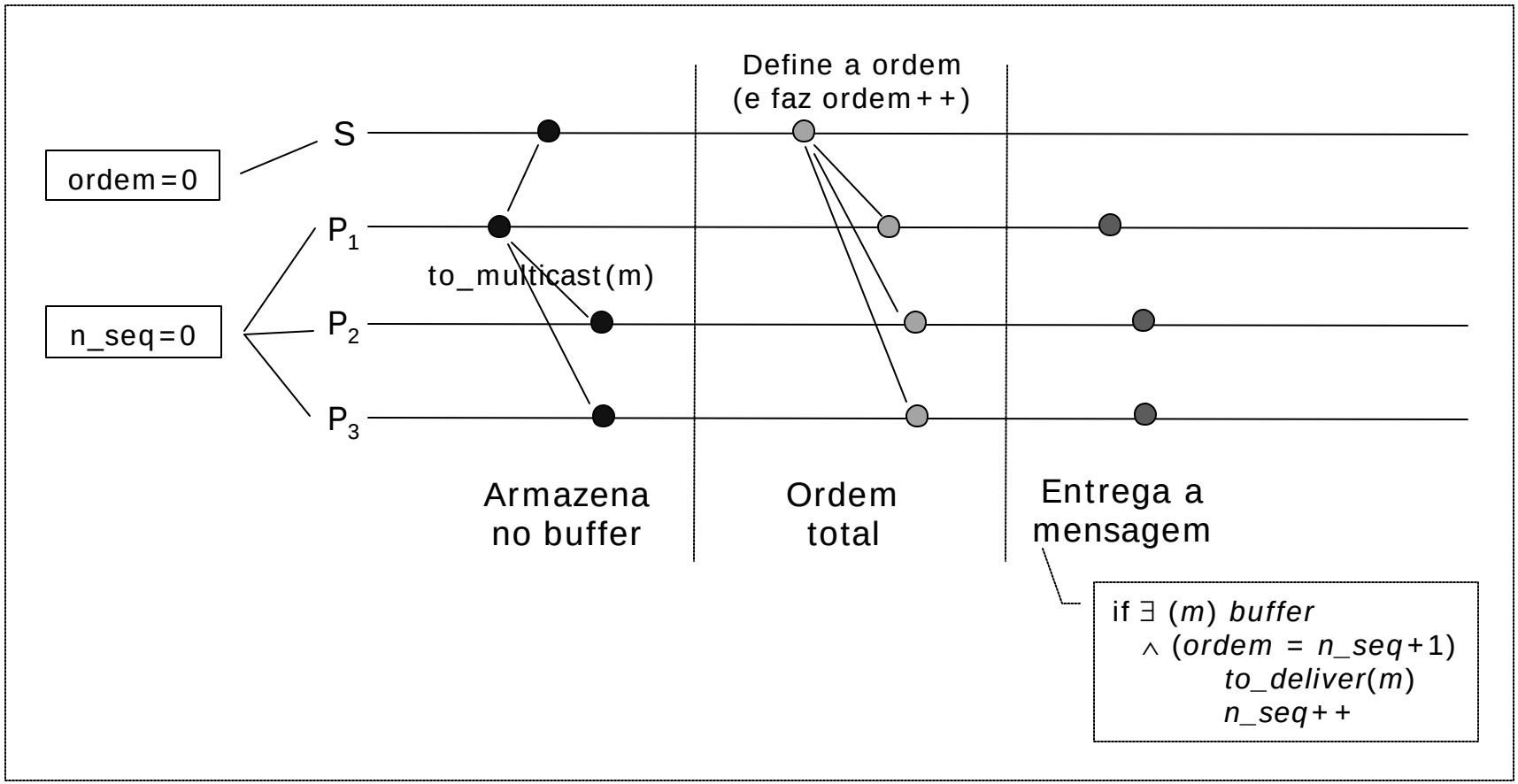
- Assegura que todos os processos receberam todas as mensagens na mesma ordem
- Operadores primitivos
  - *to_multicast*(*m*)
  - *to_deliver*(*m*)

Ordem total

- Algoritmos de ordem total
  - baseado em seqüenciador
  - baseado em consenso
- Em ambos os casos:
  - mensagens recebidas são mantidas em um buffer até que se saiba a sua ordem de entrega
  - entregam-se as mensagens com o menor nº de seqüência

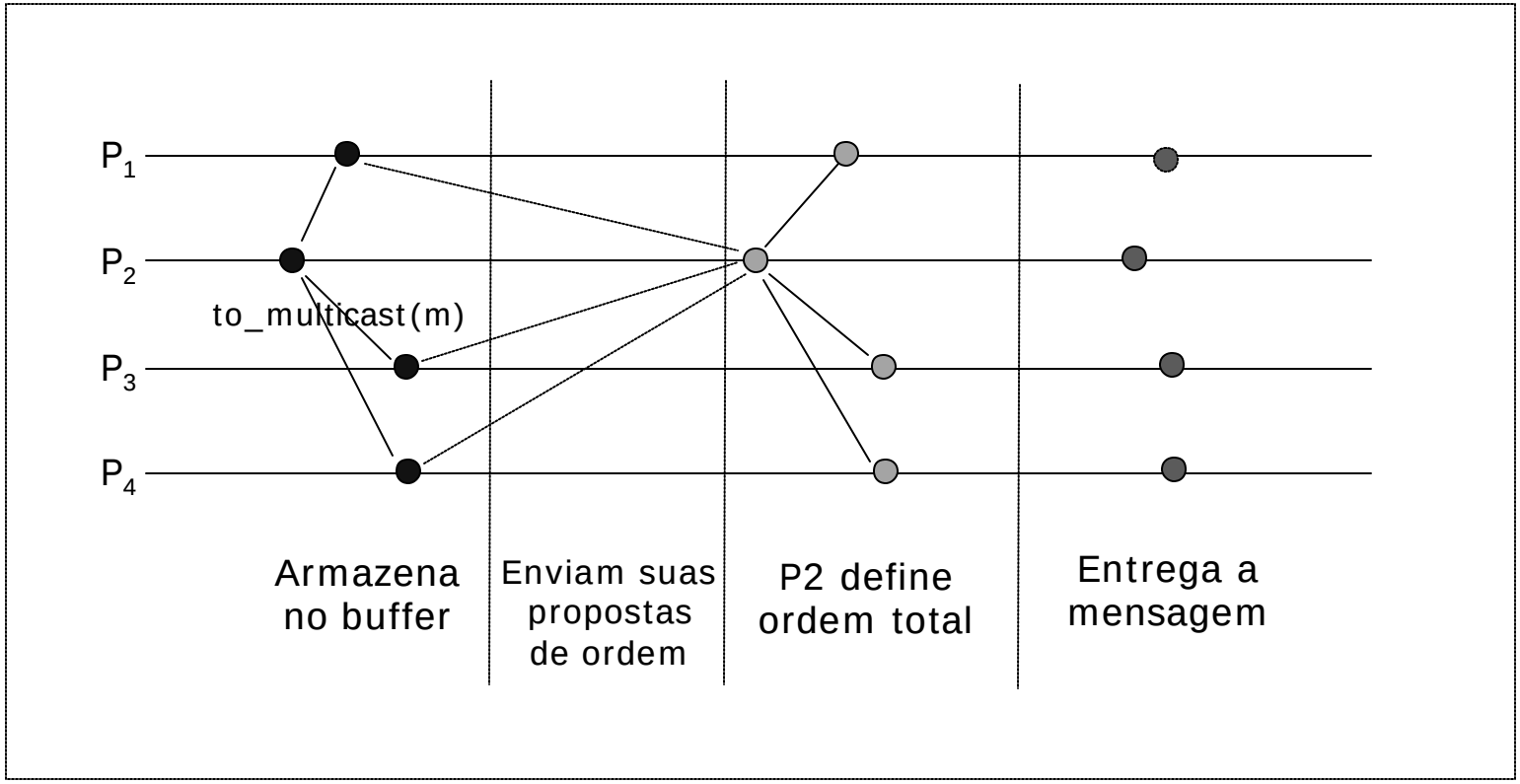
Ordem total

- Baseado em seqüenciador – visão geral



Ordem total

- Baseado em consenso – visão geral



Ordem total optimista

- Consideração optimista de que com a difusão fiável têm-se as mensagens ordenadas espontaneamente
- Baseia-se nos fatos de que:
  - alguns protocolos (como ordem total com seqüenciador) produzem uma ordem igual à ordem espontânea de alguns processos
  - em redes locais, a ordem espontânea de mensagens de todos os processos é geralmente muito similar à ordem final.

Ordem total optimista

- Vantagem: melhora a latência média de um protocolo de ordem total baseado no consenso ou em seqüenciador
- Utilização: deve-se balancear a taxa de mensagens entregues na ordem correta com o custo no caso de erro
- Operadores primitivos:
  - *opt_deliver*(*m*): entrega optimista
  - *fnl_deliver*(*m*): entrega autoritária (ordem)
- Janela optimista: tempo entre a entrega optimista e a autoritária -> aplicação pode realizar processamento optimista.

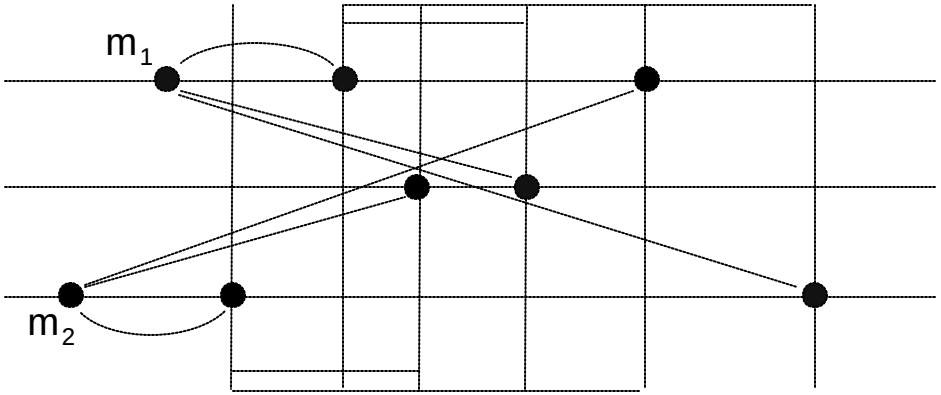
Obstáculos para ordem total espontânea

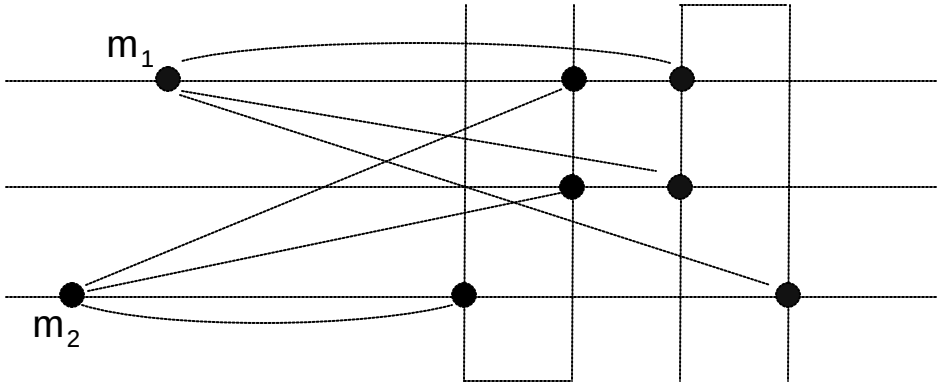
- Difícil obter alta taxa de mensagens com ordem espontânea que resulte em boa performance de aplicações optimistas (principalmente em WAN).
- Algumas razões:
  - loopback
  - pacotes perdidos por alguns processos: falha detectada e retransmissão é feita, mas outra mensagem pode chegar ao processo antes
  - mensagens recebidas em tempos diferentes: nós mais próximos aos senders

Protocolo proposto

Idéia inicial

- Quando a variação no atraso das mensagens entre qualquer par de processos é baixa, pode-se utilizar ordem total espontânea


m_1
m_2


m_1
m_2

Como o protocolo aplica essa idéia

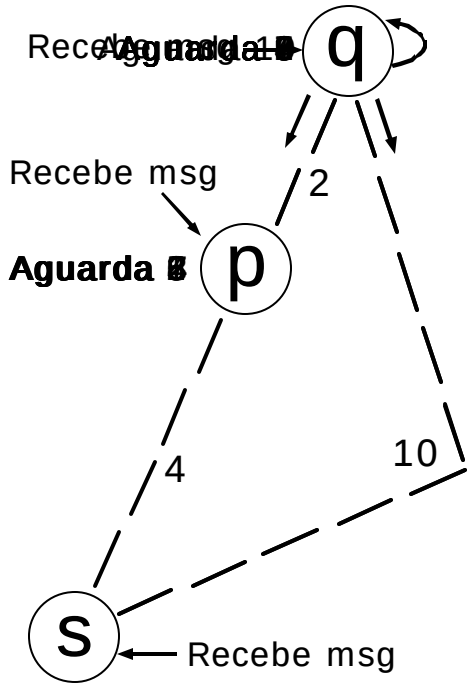
- Torna todos os processos *p* e o seqüenciador *s* eqüidistantes em relação a cada sender *q*
  - aumentar a distância de *q* para *p*: aumentar o delay que *p* impõe para *q*
  - diminuir a distância de *q* para *p*:
    - diminuir o delay que *p* impõe para *q*
    - ou, caso não seja possível, aumentar o delay de *p* para os demais processos

Como o protocolo aplica essa idéia

- Casos particulares:
  - Distância de si mesmo: cada processo utiliza a sua distância até o sequencer
  - Para o sequencer se distanciar de si mesmo (mensagens optimistas): utiliza o maior valor sugerido entre os senders
    - problema: a mensagem autoritária também será atrasada

Cálculo da distância

- Exemplo



- Ajuste da distância (delay)
  - Processo p
    - delay[p] = 4
    - delay[q] = 8
    - delay[s] = 6
  - Processo q
    - delay[q] = 10
    - delay[p] = 2
    - delay[s] = 0
  - Sequencer s
    - delay[s] = 10
    - delay[p] = 0
    - delay[q] = 0

Distância de si mesmo = 0

Distância de *q* até *s*

Opção 1 = diminuir delay[s] em *q*: não é possível, já que *q* tem delay[s] = 0

Opção 2 = distanciar *q* dos demais processos: aumentar os outros delays de *q*.

delay[s] = max(propostas_senders) − propostas_senders = max(delay) − delay[s]

Exemplo

- proposta de *p* = 8-6= 2
- proposta de *q* = 10-0 = 10

Então, *s* terá delay[s] = 10

Considerações

- sistema assíncrono composto por conjunto finito de processos seqüenciais que se comunicam através de um protocolo fiável totalmente conectado ponto-a-ponto
- processos não têm acesso à memória compartilhada ou relógio global
- processos só podem falhar por “crash” e não têm como se recuperar
- processos que nunca falham -> “corretos”
- a maioria dos processos é correto
- problema do consenso tem solução

Algoritmo

```
g ← 0                  {Global sequence nº}
l ← 0                  {Local sequence nº}
R ← ∅                  {Messages received}
S ← ∅                  {Sequence numbers}
O ← ∅                  {Messages opt-delivered}
F ← ∅                  {Messages fnl-delivered}
delay[1..n] ← 0
r_delay[1..n] ← 0      {Delays proposed to S}
```

```
procedure TO_multicast(m) do
    R_multicast(DATA(m, max(delay[]) - delay[seq]))

upon R_deliver(DATA(m, d)) do
    R ¬ R U {(m, d, now + delay[m.sender])}

upon $(m, d, t) Î R : now = t ? m ÏO ? m Ï F do
    opt_delivery(m)
    O ¬ O U {m}
    if p = seq then
        g ¬ g + 1
        R_multicast(SEQ(m, g))
        r_delay[m.sender] ¬ d
        delay[p] ¬ max(r.delay[])
```

Reliable Multicast

Sugestão de delay para *S* impor nas suas próprias msg

Tempo que o processo irá esperar para entregar a msg

Já é hora de entregar a msg

Ainda não entregou optimista

Ainda não entregou autoritária

Se for o Sequenciador

Faz reliable multicast enviando a ordem de *m*

Ajusta delay p/ impor nas suas msg optimistas

Delay de *S* p/ receber sua própria msg optimista

Algoritmo

```
g ← 0                    {Global sequence nº}
l ← 0                    {Local sequence nº}
R ← ∅                    {Messages received}
S ← ∅                    {Sequence numbers}
O ← ∅                    {Messages opt-delivered}
F ← ∅                    {Messages fnl-delivered}
delay[1..n] ← 0
r_delay[1..n] ← 0        {Delays proposed to S}
```

```
upon R_deliver(SEQ(m, s)) do
   S ¬ S U {(m, s, now)}

upon $(m, d, o) Î R : (m, l + 1, t) Î S ? m Ï F do
   fnl_delivery(m)
   if $(m', d', o') Î R : (m', l, t') Î S then
      ? ¬ (t – t') - (o – o')
      if ? > 0 then
         adjust(m'.sender, m.sender, ?)
      else
         adjust(m.sender, m'.sender, |?|)
   l ¬ l + 1
   F ¬ F U {m}
```

Mensagem entregue antes de *m*

Processo possui a próxima msg que deve ser entregue

Calcula a diferença entre:
- intervalo de recebimento do seqüencial de *m* e *m'*
- intervalo entrega otimística de *m* e *m'*

Ordem reversa ou ?Opt < ?Fnl

Ainda não foi entregue a mensagem *m'*

Diminui *delay*[*m'.sender*]

Diminui *delay*[*m.sender*]

Algoritmo

```
g ← 0                  {Global sequence nº}
l ← 0                  {Local sequence nº}
R ← ∅                  {Messages received}
S ← ∅                  {Sequence numbers}
O ← ∅                  {Messages opt-delivered}
F ← ∅                  {Messages fnl-delivered}
delay[1..n] ← 0
r_delay[1..n] ← 0      {Delays proposed to S}
```

```
procedure adjust(i, j, d) do
   v ¬ (delay[i] x a) + (delay[i] - d)x(1 - a)   Calcula novo delay para i
   if v = 0 then                                 Se v = 0, atribui v a delay[i]
      delay[i] ¬ v
   else                                          Se v < 0, não é possível diminuir delay[i]
      delay[i] ¬ 0                               zera delay[i]
      delay[j] ¬ delay[j] + |v|                  aumenta delay[j]  (distancia j)
```

Critérios de avaliação

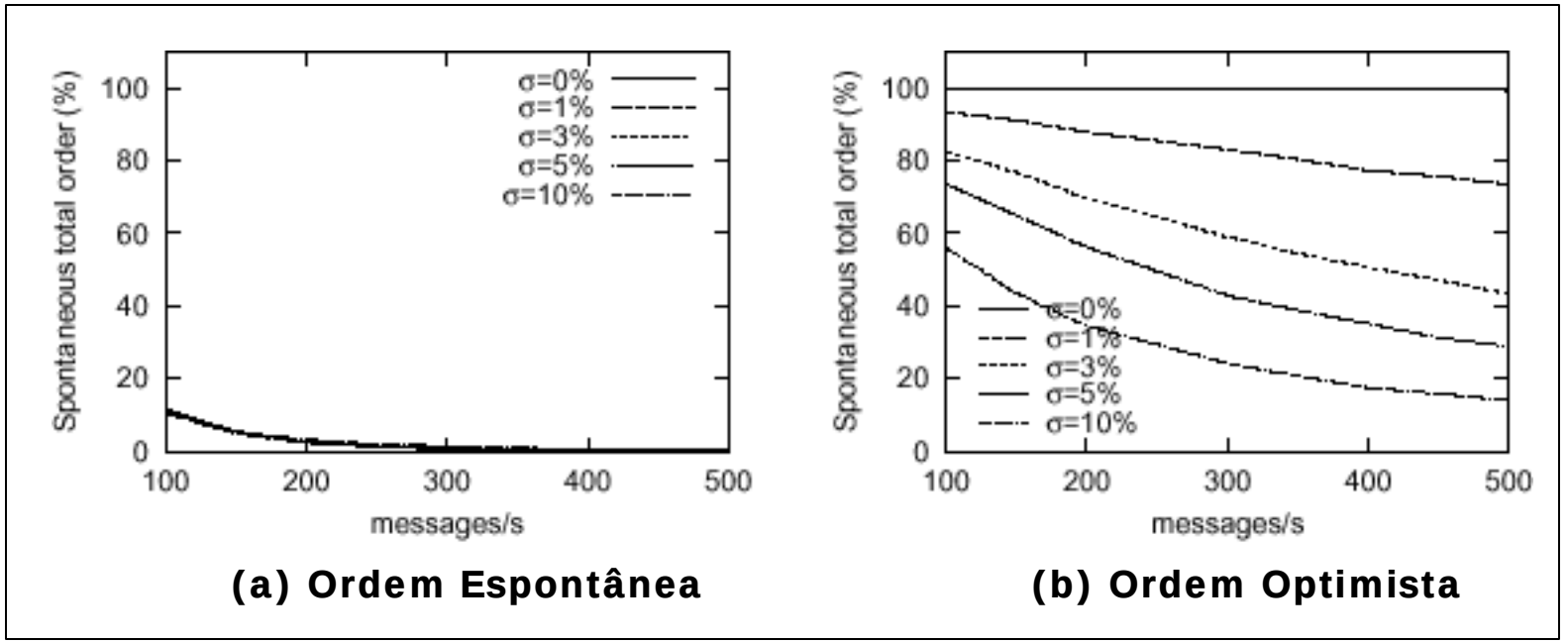
- Ordem optimista bem próxima à autoritária: comparar o algoritmo original e o modificado.
- Janela optimista suficiente para realizar processamento significativo desde que não aumente às custas da latência gerada p/ delays: gerar a latência média para um processo a partir de vários senders e a partir de si mesmo.

Teste de performance

- Simulação baseada em eventos
- Sem overhead para processamento e entrega de mensagens ou overhead de aplicação
- Rede totalmente conectada ponto-a-ponto
- Link de longa distância separa 2 grupos de processos
- Sem limite de banda
- Cada simulação roda por 100s, descartando as primeiras mensagens para estabilizar os ajustes os delays.
- a = 0,95

Teste de performance

- Mensagens entregues na ordem correta



Atrasos médios entre processos

- Dentro do grupo: 20ms
- Fora do grupo: 40ms
- Si mesmo: 0ms

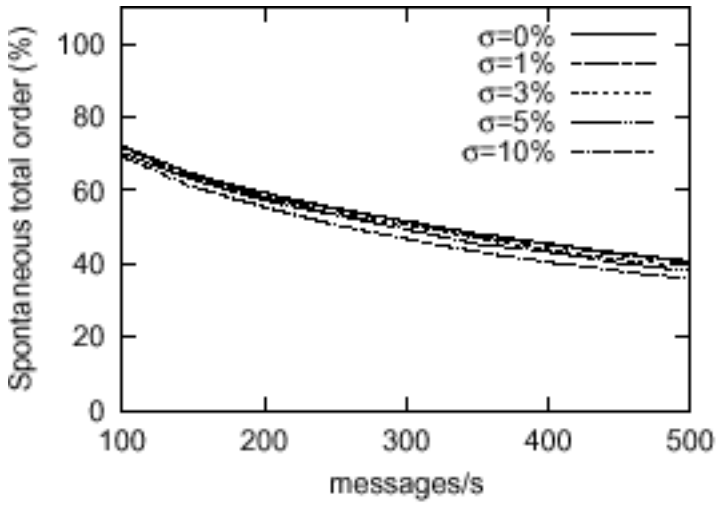
S = Variação de atraso

Obs: na Internet, a variação no atraso das transmissões é menos que 10% para pacotes grandes e menos de 1% para pequenos.

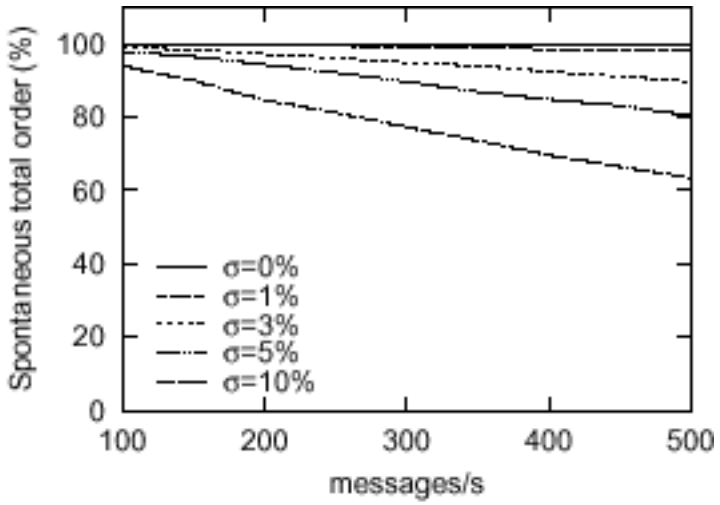
Teste de performance

- Mensagens entregues na ordem correta

Database state machine – batches de mensagens de tamanho 2



(a) Ordem Espontânea



(b) Ordem Optimista

Teste de performance

- Latência end-to-end e Janela optimista (ms)

	Sender	O próprio nó		Todos	
	Compensação	sem	com	sem	com
Sequencer	latência	0	41,4	28,5	32,3
	janela opt.	0	0	0	0
Nó próximo a *s*	latência	40,1	40,2	48,8	52,6
	janela opt.	40,1	21,8	20,3	20,2
Nó distante de *s*	latência	80,6	80,8	69,3	73,0
	janela opt.	80,6	27,3	42,0	24,6

Sem compensação de delay, a janela optimista é igual à latência

Aumento na latência final próximo para todos os casos (3,7-3,8 ms)

Validação dos resultados

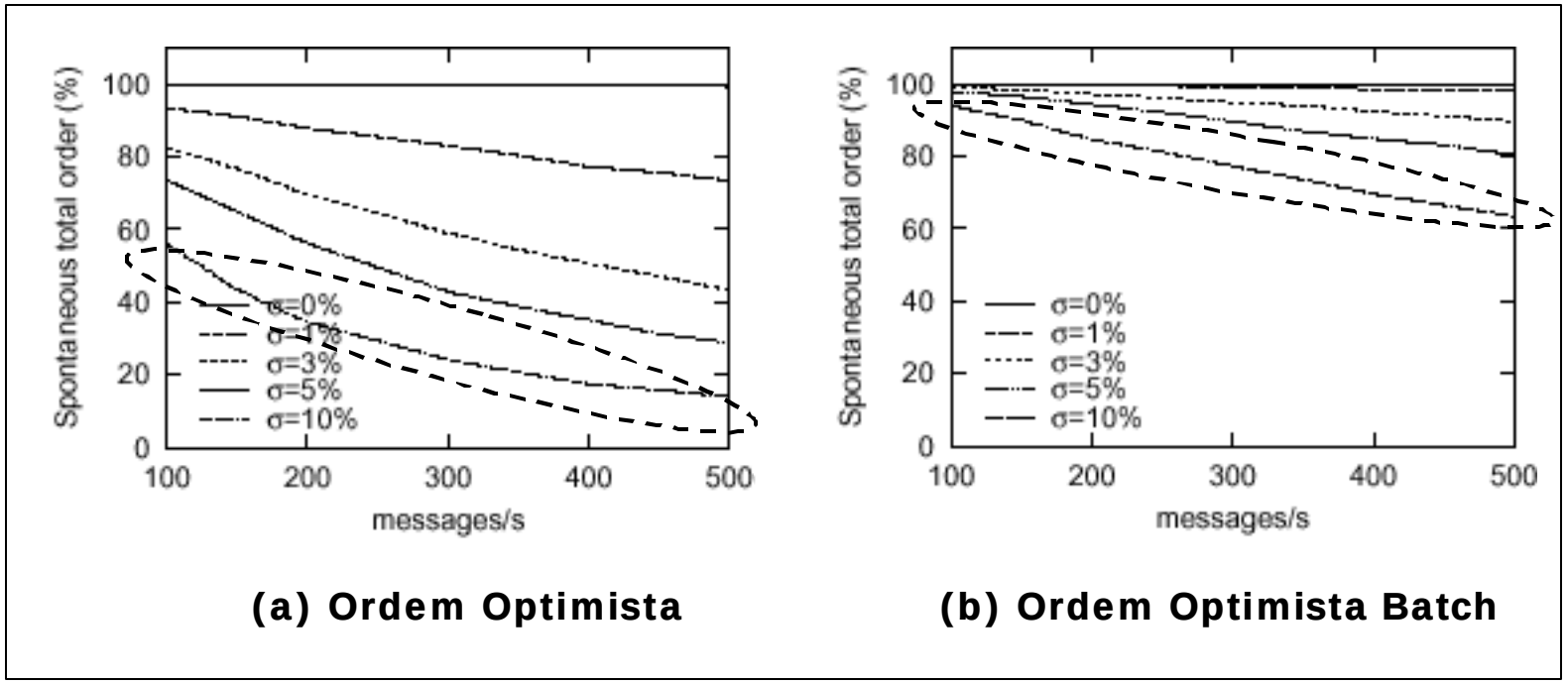
- Implementação de protocolo multicast fiável sequencial
  - Utiliza UDP ponto-a-ponto
  - Recuperação de erro realizada pelo receptor quando verifica que faltou uma mensagem (algoritmo gossip)
  - Visão do grupo e sincronia na vista utilizam consenso
  - Roda sobre rede simulada
- Entre cada par de nó: de 8 a 11 saltos
- Links redundantes: rotas diferentes; variação nos atrasos de transmissão
- Tráfego realista HTTP: 80 clientes e 20 servidores igualmente espalhados pela rede
- Atraso: de 20 a 100ms
- a = 0,95

Validação dos resultados

- WAN: backbone 2 roteadores conectados por link de longa distância de 34Mbps, os quais se conectam ao roteador principal de cada LAN.
- LAN: backbone de 3 roteadores p/ tolerância a faltas, 2 conectados ao principal. Os servidores de aplicação conectam-se aos roteadores de secção, e estes aos roteadores do backbone para tolerância a faltas.

Validação dos resultados

- Mensagens entregues na ordem correta



Latência efectiva

- g = tempo de processamento da msg
- l = latência da entrega autoritária
- w = tamanho da janela optimista
- r = taxa de msg optimistas corretas

- Se $g \leq w$
  - se a entrega optimista estiver correta
    latência efectiva = l
  - senão latência efectiva = $l + g$
  - Latência média = $rl + (1\text{-}r)(l+g)$
- Senão, se $g \geq w$:
  - se a entrega optimista estiver correta
    latência efectiva = $l + g - w$
  - senão latência efectiva = $l + g - w + g$
  - Latência média = $r(l+g\text{-}w) + (1\text{-}r)(l+2g\text{-}w)$

Conclusões

- Facilmente aplicado ao algoritmo de OT com seqüenciador que é comprovadamente correto em sistemas assíncronos -> assegura a robustez da solução proposta.
- Modificação no algoritmo original: apenas compensa as diferenças nos atrasos de transmissão, com custo computacional mínimo e sem mensagens adicionais.
- Provavelmente o único algoritmo com latência end-to-end efectiva menor que o atraso de ida e volta da mensagem.