Serviço de timeline descentralizado

Sistemas Distribuídos em Larga Escala

Ana Rodrigues (a78763), Armando Santos (a77628), and Cláudia Correia (a77431)

Universidade do Minho, Braga, Portugal

Resumo Este documento diz respeito ao projeto proposto na Unidade Curricular de Sistemas Distribuídos em Larga Escala da Universidade do Minho, que consiste na criação de um serviço de *timeline* descentralizado.

1 Introdução

Este projeto tem como finalidade a criação de um serviço de *timeline* descentralizado, à semelhança do Facebook, Twitter ou Instagram. Este tipo de serviços consiste na partilha de publicações entre diferentes utilizadores, sendo que em cada momento existe um grande número de publicações a circular pela rede. À medida que os utilizadores vão aderindo ao serviço, o número de publicações aumenta radicalmente. Tendo em conta este aspeto, uma arquitetura na qual existam servidores centralizados torna-se bastante ineficiente, visto que podem ficar rapidamente sobrecarregados, comprometendo a disponibilidade do sistema. Nestes casos, uma boa solução é a arquitetura *Peer-to-peer*, na qual os utilizadores desempenham também a função de servidor. Adotando uma arquitetura *Peer-to-peer*, no caso de um serviço de *timeline*, os utilizadores irão não só realizar as suas funções básicas, como também armazenar publicações e disponibilizá-las a outros utilizadores, daí ser um serviço descentralizado.

Neste contexto, cada utilizador pode fazer publicações e ver as publicações dos utilizadores que subscreveu, sendo que estas lhe são apresentadas por uma determinada ordem temporal. Para além disto, cada utilizador vai armazenando todas as publicações que vão aparecendo na sua *timeline*, podendo posteriormente encaminhá-las. Apesar disto, todas as publicações que não foram feitas pelo próprio e estão a ser armazenadas, são guardadas e encaminhadas durante um período de tempo estipulado.

Apesar da arquitetura *Peer-to-peer* assegurar elevada disponibilidade, é necessário que a comunicação entre os diferentes utilizadores seja feita de forma eficiente, caso contrário esta escolha pode ter um impacto negativo no serviço. Sendo assim, a disposição da rede e a forma como a informação se propaga é um dos pontos cruciais para a criação de um serviço descentralizado com bom desempenho.

Posto isto, nas secções que se seguem, serão apresentadas as considerações feitas e decisões tomadas na implementação de um serviço de *timeline* descentralizado. Serão abordados temas como a arquitetura do sistema, os componentes que o constituem e a interação entre eles. Numa fase final será também apresentado todo o processo de implementação do serviço.

2 Arquitetura

Estando perante um serviço de *timeline*, é de prever que serão inúmeras as mensagens propagadas pela rede. Desde mensagens de subscrição até mensagens relativas a publicações, a informação que circula na rede em cada momento é imensa e, portanto, este representa o maior obstáculo para a escalabilidade do serviço.

Imaginando a rede como um grafo, numa primeira fase a solução mais ingénua será assumir que cada utilizador deverá estar ligado aos seus subscritores, para que lhes possa transmitir diretamente as publicações que fizer. No entanto, seguindo esta linha de raciocínio, também o próprio utilizador, enquanto subscritor, deverá estar ligado a outros utilizadores. Para além disto, não nos podemos esquecer de que, sendo esta uma arquitetura *Peer-to-peer*, cada utilizador deve também estar ligado aos restantes subscritores dos utilizadores que subscreve, para lhes pedir publicações caso o próprio utilizador esteja *offline*. Com esta abordagem facilmente nos apercebemos de que a rede vai ficar completamente desestruturada e confusa, e de que existirá um peso adicional relativo ao registo de quais os vizinhos de cada nodo e qual a relação existente entre eles. Abaixo estão representadas as conexões (linhas a tracejado) que resultam apenas da subscrição entre o *utilizador A* e o *utilizador B*.

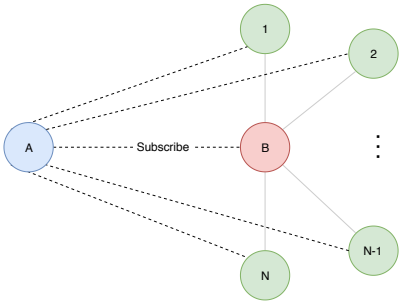

Figura 1: Conexões resultantes de uma subscrição

Tendo em conta que esta estratégia seria bastante ineficiente, optou-se por adotar um método baseado na teoria do *Super-Peer*. Por conseguinte, em vez de existirem conexões entre os próprios utilizadores baseadas nas relações do serviço de *timeline*, estes conectam-se a um nodo com "maior poder", denominado **Superuser**. Assim, para propagar uma mensagem os utilizadores necessitarão apenas de comunicar a informação ao seu *Superuser*, e este tratará de encaminhar a mensagem para outro utilizador, ou para outro *Superuser*, caso seja necessário. Desta forma, cada *Superuser* será responsável por um domínio dos utilizadores, e em vez de existirem trocas de mensagens constantes entre os diferentes utilizadores do sistema, a comunicação é intermediada pelos *Superusers*.

Em contraste com a primeira abordagem descrita, com a adição de *super-peers*, para além da rede não ser tão sobrecarregada com mensagens, os utilizadores também não precisam de manter estado relativo às suas conexões dentro da rede, visto que passam apenas a estar conectados a um *superuser*.

Outro aspeto pertinente é a ligação de um *peer* à rede. Quando um utilizador se regista ou faz *login*, tem de existir algum ponto central que seja capaz de o integrar na rede. A este interveniente foi dado o nome de **Central**. Para além de incluir os nodos na rede, a *Central* gere os *Superusers* existentes, tal como veremos mais à frente, e mantém registo dos mesmos para conseguir atribuí-los a cada novo utilizador que se liga ao serviço. Desta forma, a *Central* possui apenas a informação mínima necessária para desempenhar as suas funções, tornando assim o serviço o máximo descentralizado possível, sendo este uma das principais preocupações na implementação deste projeto.

A figura abaixo descreve a arquitetura final do sistema, baseada na descrição feita acima.

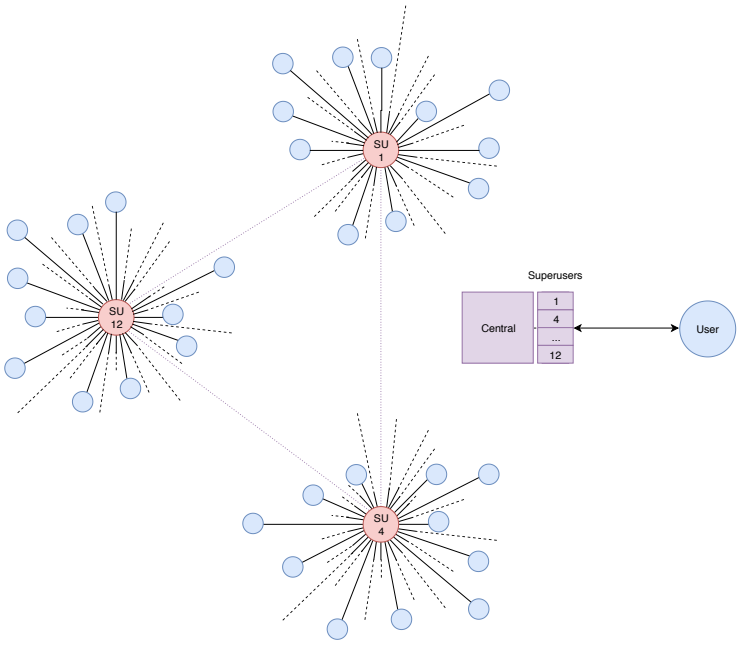

Figura 2: Arquitetura do sistema

3 Componentes

Com o intuito de começar a delinear o processo de implementação, foi estudado o problema de forma a identificar os papeis distintos que necessitam de ser desempenhados no sistema, de forma a alcançar os objetivos de um serviço de *timeline*.

3.1 User

À luz de vários serviços bastante conhecidos como os enunciados em 1, podemos catalogar o serviço de *timeline* em questão como uma rede social onde os utilizadores e as relações entre os mesmos são o ponto fulcral do problema. Sendo um dos objetivos deste projeto implementar o serviço de forma descentralizada (arquitetura *Peer-to-peer*), faz sentido que a noção de *peer* seja incorporada nos utilizadores do serviço. Tendo dito isto, um dos componentes identificados é o que representa os utilizadores do sistema, assim como os *peers* da arquitetura, e é denominado User.

3.2 Follower e Followee

Um dos conceitos mais básicos numa rede social é o chamado *círculo de amigos* que representa as relações entre os utilizadores. A relação mais simples existente num círculo de amigos é a *follows* (cuja inversa é a *is followed by*) e é a relação que, por conveniência, determina o papel que cada utilizador desempenha no serviço de *timeline*. Com base nesta relação, por questões de organização e simplificação, foram identificados os componentes Follower e Followee. O *Follower* representa a faceta de subscritor, que consome publicações que vão sendo feitas, armazena essas mesmas publicações e encaminha-as a outros *Followers*. Em contrapartida, o *Followee* representa o papel do utilizador que faz publicações, propagando-as para os seus *Followers* e guardando-as localmente.

3.3 Central

Pensando na rede social como um serviço *Peer-to-peer*, um dos maiores desafios foi o de obter uma noção de individualidade, ou seja, uma noção de identidade associada a cada utilizador que, no mínimo, garante a unicidade de cada um no sistema. Para além de gerir os *Superusers* e integrar os utilizadores na rede, tal como foi descrito em 2, a Central garante também que cada utilizador possui uma entidade única.

3.4 Superuser

De acordo com o que foi dito em 2, uma vez que não é desejável que o sistema, a partir de um certo número de utilizadores, comece a deteriorar o seu desempenho, é indispensável que este seja escalável. Com o objetivo de garantir esta qualidade, outro dos componentes do sistema é o Superuser. É importante mencionar que o *Superuser* não passa de um *User* normal, com uma responsabilidade acrescida.

4 Funcionamento geral

Relativamente ao funcionamento geral do sistema, existe um fio condutor que é mantido na interação entre os diferentes componentes do mesmo.

Tomando como ponto de partida a ligação de um *User* ao serviço, nesta fase o *User* comunica à *Central* a sua vontade de fazer *sign in* ou *sign up*, e a *Central* trata de autenticar o *User*, tendo em mente que não devem existir *usernames* repetidos. Depois de autenticado, a *Central* informa o *User* de qual o *Superuser* a que o mesmo se deve conectar.

Depois de estar conectado a um *Superuser*, o *User* pode desempenhar funções tanto de *Follower* como de *Followee*, no entanto, numa fase inicial deve solicitar publicações aos seus *Followees* (ou aos *Followers* dos seus *Followees*, caso os próprios não estejam *online*). Após isto, o *User* vai efetuando o seu funcionamento normal de publicar conteúdo, encaminhar e processar publicações.

No decorrer do processo os *Users* comuns auto-intitulam-se *Superusers* caso cumpram com algumas condições, e comunicam esta "subida de posto" à *Central*. Na falta de *Superusers*, a *Central* também é capaz de promover *Superusers*. Caso algum dos *Superusers* fique *offline*, a *Central* é responsável de fazer a transição dos *Users* associados a esse nodo para outro *Superuser*.

5 Tecnologias utilizadas

As tecnologias eleitas para a implementação do serviço que tem vindo a ser detalhado foram Java e o Spread Toolkit. Esta escolha será devidamente descrita e justificada de seguida.

5.1 Java

Sendo esta uma linguagem *multi-threaded*, o que já por si é bastante vantajoso neste contexto, Java possui diversas *frameworks* e bibliotecas úteis para sistemas distribuídos, tornando-se assim uma boa opção para a criação de um serviço descentralizado.

5.2 Spread Toolkit

Tal como foi dito anteriormente, na presença de uma arquitetura *Peer-to-peer*, a comunicação entre os pontos da rede deve ser o mais eficiente possível. Para tratar este problema optou-se por utilizar o Spread Toolkit, que fornece um sistema de comunicação em grupo confiável e de alto desempenho. O facto deste *toolkit* providenciar um sistema de comunicação escalável é extremamente importante, visto que é previsível que serviços de *timeline* escalem rapidamente com a adesão de novos utilizadores, e é necessário que as tecnologias utilizadas consigam suportar este crescimento. Para além do Spread garantir resiliência a falhas em redes locais e de longa distância, e fornecer a possibilidade de manter uma ordenação causal das mensagens, o seu sistema de comunicação em grupo fornece uma abstração bastante adequada para implementar a noção de *círculo de amigos*. [2]

Em termos de funcionamento, o Spread Toolkit disponibiliza primitivas de comunicação em grupo que permitem, entre outros, entrar e sair de um grupo. Para além disto, através do envio de mensagens de *membership*, este *toolkit* fornece informação acerca da entrada/saída de um membro do grupo.

A gestão da comunicação entre os diferentes grupos é feita pelo Spread através de um *daemon* ao qual os utilizadores se conectam. É possível tanto cada utilizador correr o seu próprio *daemon*, como vários utilizadores ligarem-se a um único *daemon*. Assim, independentemente dos membros de cada grupo estarem a correr na mesma máquina ou não, os *daemons* do Spread comunicam entre si, de forma a controlarem a troca de mensagens relativas a diferentes grupos.

6 Implementação

De forma a descrever o sistema da melhor forma, de seguida serão apresentados, de forma sucinta, alguns dos pormenores mais relevantes no que concerne à implementação do serviço. Tal como foi dito anteriormente, este projeto foi realizado com recurso a Java, e ao Spread Toolkit, logo, nas secções seguintes será utilizado vocabulário referente a estas tecnologias.

6.1 Arquitetura

De forma a cumprir com a arquitetura proposta na figura 2, e tendo em conta o funcionamento do Spread Toolkit, optou-se por adotar a seguinte estratégia:

- cada *Superuser* corre o *daemon* do Spread;
- cada *User* conecta-se ao *daemon* do seu *Superuser*.

Deste modo, cada *User* relata as suas ações ao *daemon* do seu *Superuser*, e este gere o que é necessário, e comunica com outros *Superusers* (outros *daemons*) apenas caso as ações realizadas pelos seus *Users* digam também respeito a outros *Users* que não estejam conectados a si. Com esta abordagem, os *Superusers* adotam então o papel de *super-peers* da rede, aos quais os *Users* se conectam, minimizando assim a troca de mensagens entre os diferentes utilizadores do serviço.

Comparativamente com a solução na qual todos os utilizadores correm o *daemon*, neste caso, sempre que fosse, por exemplo, enviada uma mensagem para um grupo, todos os *daemons* teriam de ser contactados, de forma a saber se aquele utilizador era um membro do grupo ou não. Isto é, a mensagem teria de ser propagada por todos os elementos da rede (*multicast*), o que tornaria o processo muito menos eficiente. Com

isto, podemos concluir que existindo uma boa relação entre o número de *Users* e o número de *Superusers*, a escalabilidade do serviço é bastante alta [3].

Outro dos componentes da arquitetura presente na figura 2 é a *Central*. Este componente representa uma máquina centralizada independente de todas as outras, que corre o seu próprio *daemon*, e que se encontra no grupo de comunicação `centralGroup`. Desta forma, ao entrar na rede, os utilizadores devem comunicar com a *Central* para se conseguirem autenticar, visto que é este componente que mantém as credenciais de todos os utilizadores do serviço.

6.2 Ciclo de vida de um User

Assim que um *User* se autentica, a *Central* atribui-lhe um *Superuser*, ao qual o mesmo se deve conectar. A partir deste momento, o utilizador passa a executar as tarefas que lhe competem como *Follower* e *Followee*.

Enquanto *Followee* este junta-se ao seu grupo de seguidores, que é identificado por `<username>Group`, sendo que `<username>` corresponde ao seu nome de utilizador. Caso efetue uma publicação, esta vai ser enviada em *multicast* para todos os membros do seu grupo, isto é, para todos os seus *Followers* (subscritores). Enquanto *Follower*, o utilizador junta-se aos grupos dos seus *Followees* e envia uma mensagem em *multicast* para cada um dos grupos a requisitar uma atualização da informação que possa ter perdido enquanto esteve ausente.

Quando um *User* pretende desconectar-se do serviço, este envia uma mensagem à *Central* a notificá-la, e de seguida sai de todos os grupos dos quais era membro, e desconecta-se do *daemon* ao qual estava ligado.

6.3 Subscrição de Users

Para um *User* ser capaz de subscrever outro basta simplesmente entrar no grupo desse *Followee*. Analogamente, para um *User* deixar de seguir outro apenas tem que sair do respetivo grupo. Ao entrar no grupo do *Followee*, a primeira coisa que o *Follower* deve fazer é pedir informação relativa às publicações do *Followee* em questão. Nesta situação o *Follower* pode-se deparar com dois casos: o *Followee* estar *online* ou estar *offline*. Estas duas situações podem ser detetadas analisando a mensagem de *membership* proveniente do facto de um novo utilizador se ter juntado ao grupo. Examinando a lista de membros presente na mensagem de *membership*, é possível verificar se o *Followee* desse grupo está presente ou não. Se estiver presente na lista significa que está *online*, então o utilizador contacta diretamente o *Followee*, pedindo-lhe as suas publicações. Caso não esteja na lista, é contactado um membro aleatório da lista, isto é, um outro *Follower* do *Followee* em questão. É importante mencionar que neste último cenário, o utilizador corre o risco de não obter a informação mais atual, assim, nos casos em que isto acontece, quando o *Followee* ficar *online*, o utilizador vai-lhe pedir as publicações mais recentes.

6.4 Ordem causal das publicações

Enquanto um utilizador está *online*, todas as publicações são-lhe transmitidas segundo uma ordem causal. Esta característica é conseguida recorrendo ao Spread, que permite garantir propriedades FIFO às mensagens e ordená-las causalmente, conforme definido por Lamport [1].

Em contrapartida, quando um utilizador faz *sign in*, ele vai tentar recuperar, a partir de outros utilizadores, as publicações que foram feitas no período em que esteve ausente. Nesta fase, não é possível garantir uma ordem causal das publicações visto que estas por vezes são fornecidas por *Followers*, em vez do próprio *Followee*. Para contornar estas situações, ao ser feita uma publicação, o autor marca-a com um *timestamp*. Assim, ao receber as publicações antigas, elas são apresentadas ao utilizador que acabou de fazer *sign in* segundo o *timestamp* que possuem. Esta abordagem pode nem sempre ser a mais correta, visto que as publicações podem não ser apresentadas segundo a ordem pela qual foram realizadas.

6.5 Armazenamento das publicações

Embora cada utilizador guarde as suas próprias publicações, este vai guardar também as publicações dos utilizadores que subscreveu. No entanto, enquanto a sua informação é guardada localmente e de forma persistente, a informação que mantém relativamente a outros é efémera. Sendo assim, ao fazer *sign in*, as publicações guardadas que são relativas a outros utilizadores vão ser filtradas. Sempre que um utilizador se

autenticar no serviço de *timeline*, todas as publicações de outros utilizadores que forem mais antigas que um determinado período de tempo x (p.e. um mês) vão ser eliminadas, garantindo assim que apenas as publicações mais recentes se mantêm guardadas.

6.6 Gestão de Superusers

Numa fase inicial, ou numa situação em que não existam *Superusers online*, é a *Central* que toma a iniciativa de promover um *User* a *Superuser*. Isto deve-se ao facto de ser necessário existir sempre pelo menos um *Superuser online* no sistema, visto que caso contrário os *Users* não teriam um *daemon* ao qual se conectarem. Dado isto, a *Central* necessita de guardar informação acerca do estado (*online* ou *offline*) de cada utilizador.

Durante o funcionamento normal do sistema, é o próprio *User* que se auto-intitula *Superuser*, tal como já foi mencionado em 4. Esta promoção é feita com base no tempo médio que o utilizador está *online*. Cada utilizador armazena de forma persistente informação relativa ao seu *uptime* médio, para que, sempre que fizer *sign in* possa decidir se será um *Superuser* ou não. No entanto, os utilizadores não se promovem apenas a *Superusers* no momento de *sign in*. Ao entrarem no sistema, é lançada uma *TimerTask* que fica ativa passado um determinado período de tempo. Caso o utilizador fique *online* durante todo esse período de tempo, a *TimerTask* promove-o a *Superuser* automaticamente.

Por esta ordem de raciocínio, existem quatro situações nas quais um *User* é promovido a *Superuser*:

- não existem utilizadores *online*, então o próximo utilizador a fazer *sign in* é automaticamente promovido a *Superuser* pela *Central*;
- o único *Superuser* que está *online* vai fazer *sign out*, então a *Central* promove automaticamente um dos *Users* que estão *online* a *Superuser*;
- ao fazer *sign in*, o *uptime* médio do utilizador é superior a um determinado período x, então este auto-intitula-se *Superuser*;
- um utilizador fica *online* mais do que um determinado período x e auto-intitula-se *Superuser*.

Posto isto, para controlar o seu *uptime*, cada utilizador armazena duas variáveis de forma persistente, `averageUpTime` e `totalSignIns`, que representam o *uptime* médio e o número total de vezes que já fez *sign in*, respetivamente. Para além destas, cada utilizador mantém também as variáveis `startTime` e `endTime`, para conseguir calcular o tempo que esteve *online* durante uma sessão. Desta forma, no final de cada sessão, a variável `totalSingIns` é incrementada e, posteriormente, a `averageUpTime` é atualizada segundo a seguinte fórmula:

$$averageUpTime = \frac{(averageUpTime \times (totalSignIns - 1) + (endTime - startTime))}{totalSignIns}$$

Outro dos aspetos importantes é o que acontece aos *Users* que estão conectados a um *Superuser* quando este faz *sign out*. Neste caso assumiu-se que estamos perante um modelo *fail-stop*, no qual os *Superusers* notificam a *Central* quando fazem *sign out*. Assim, quando a *Central* recebe a informação de que um *Superuser* vai fazer *sign out*, ela indica imediatamente a todos os *Users* a ele conectados, um novo *Superuser* ao qual se devem conectar. Para tornar este processo eficiente, quando um *User* se conecta a um *Superuser*, para além de se ligar ao seu *daemon*, vai também entrar no grupo `<username>SuperGroup`, onde `<username>` representa o nome de utilizador do *Superuser*. Com esta estratégia, não se corre o risco do *Superuser* fazer *sign out* antes de todos os seus *Users* estarem conectados a outro *daemon*, visto que este só vai desconectar efetivamente quando vir que ele é o único elemento do seu *super group*.

7 Conclusões e Trabalho Futuro

De uma forma geral consideramos que o projeto foi concluído com sucesso, dado que foi efetivamente criado o serviço de *timeline* descentralizado e foram alcançados todos os objetivos previstos inicialmente. É importante enfatizar que a utilização do Spread Toolkit simplificou bastante a concretização do serviço de *timeline*, principalmente no que toca a subscrições, visto que esta ação tornou-se tão simples quanto entrar/sair de um grupo de comunicação. Também com um simples método Java, foi possível garantir a

ordem causal das mensagens, fazendo com que estas vão sendo apresentadas ao utilizador que está *online*, pela ordem correta pela qual foram efetuadas. Apesar disto, para ser possível garantir a comunicação entre *daemons* que correm em máquinas diferentes, o Spread necessita que sejam alterados ficheiros de configuração e que se inicie efetivamente os *daemons* nas máquinas com o papel de *Superusers*. Este procedimento acabou por não ser implementado, no entanto, futuramente caso se pretenda utilizar o serviço em grande escala, facilmente se conseguem realizar estas alterações.

Outra modificação que poderia ser aplicada ao produto final deste projeto, era aplicar redundância à *Central*, prevenindo assim a existência de um ponto único de falha. Neste momento, embora possa existir mais do que uma *Central* na rede, todas vão responder aos pedidos dos clientes porque a mensagem é enviada em *multicast* para o `centralGroup`. No entanto, poderia ser implementado algum mecanismo que permitisse encaminhar o pedido em *unicast*, por exemplo para a *Central* mais próxima do utilizador, tornando assim o serviço mais eficiente. Uma vez que a *Central* guarda as credenciais dos utilizadores e guarda informação relativa aos *Superusers*, seria necessário implementar um protocolo de replicação entre as diferentes *Centrais*.

Embora o serviço criado esteja perfeitamente funcional e robusto, existem alguns melhoramentos relativamente simples que poderiam ser realizados, nomeadamente:

- na promoção do *Superuser*, ter também em conta parâmetros como a largura de banda, o CPU, a memória, entre outros;
- implementar algum mecanismo de recuperação das publicações, visto que cada utilizador armazena as suas publicações na máquina local, e caso faça *sign in* noutra máquina já não tem acesso a essa informação;
- configurar automaticamente (ou permitir ao utilizador configurar) os períodos de tempo utilizados para limpar as publicações antigas que subscreveu;
- configurar automaticamente o período de tempo que um *User* deve estar *online*, para ser considerado um *Superuser*.

Referências

1. LLC, S.C.: Spread 4.0.0 c api (2006), `http://www.spread.org/docs/spread_docs_4/docs/message_types.html`
2. LLC, S.C.: The spread toolkit (2016), `http://www.spread.org/`
3. Yair Amir, Claudiu Danilov, M.M.A.J.S., Stanton, J.: The spread toolkit: Architecture and performance (2004)