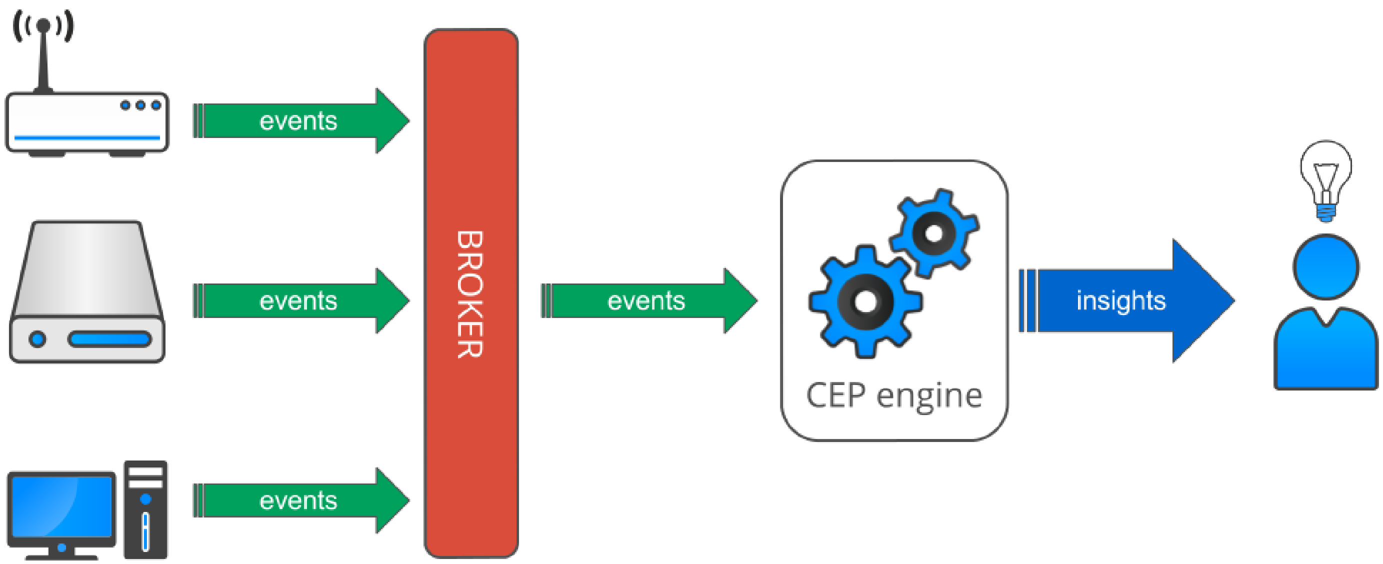
Analisando e reduzindo grandes fluxos de dados em tempo real

Como um pouco de álgebra pode ajudar

Juan Lopes - Intelie
QCon SP - 30 de Agosto de 2013

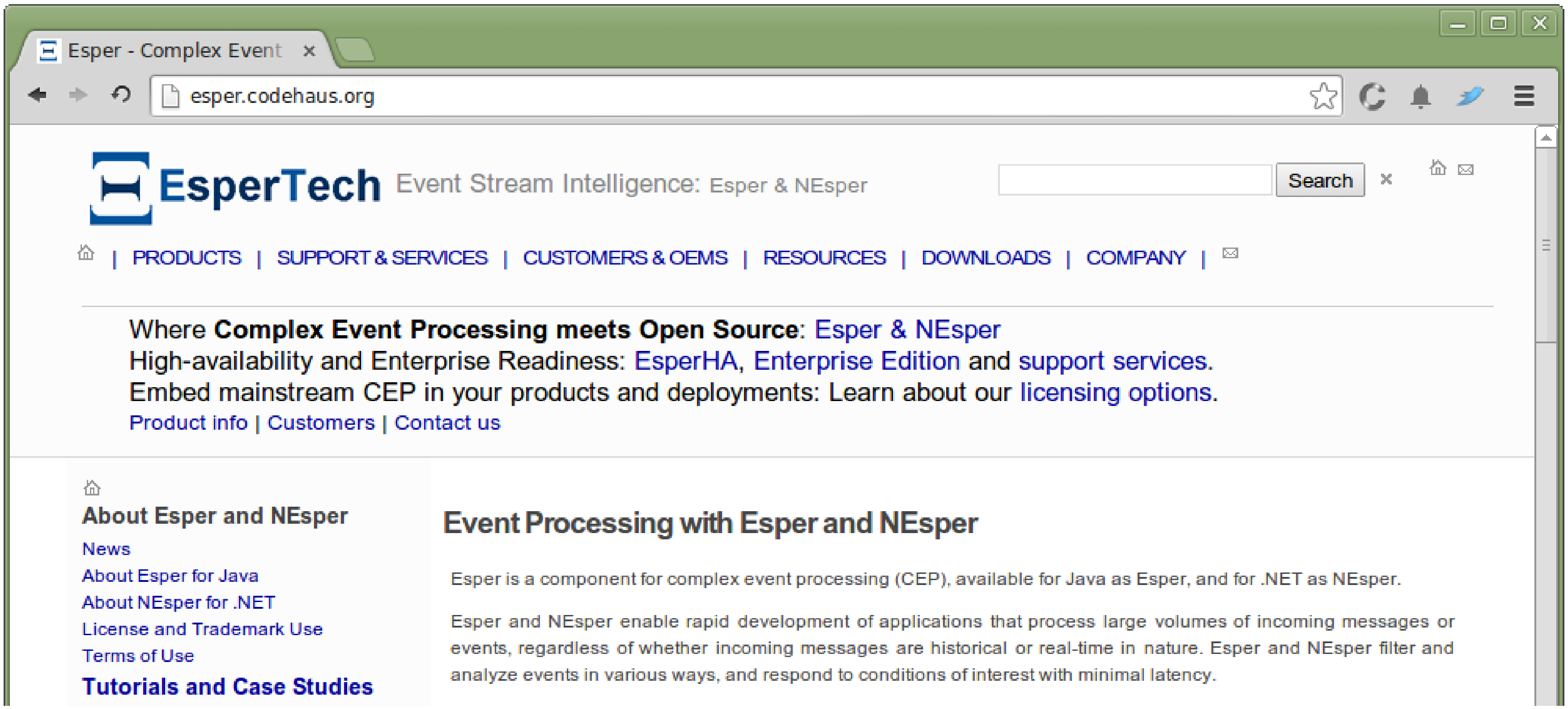
Complex Event Processing (CEP)

Para monitoração da infra de TI



Esper

Componente de CEP open-source (GPL v2)



Que linguagem é essa?

Claro que é uma pegadinha.

SQL?

```
select * from StockTick
```

HEIN?

```
select * from StockTick(symbol='IBM').win:time(30 sec)
```

EPL!

```
select a.custId, sum(a.price + b.price)
from pattern [every a=ServiceOrder ->
    b=ProductOrder(custId = a.custId) where timer:within(1 min)].win:time(2 hour)
where a.name in ('Repair', b.name)
group by a.custId
having sum(a.price + b.price) > 100
```

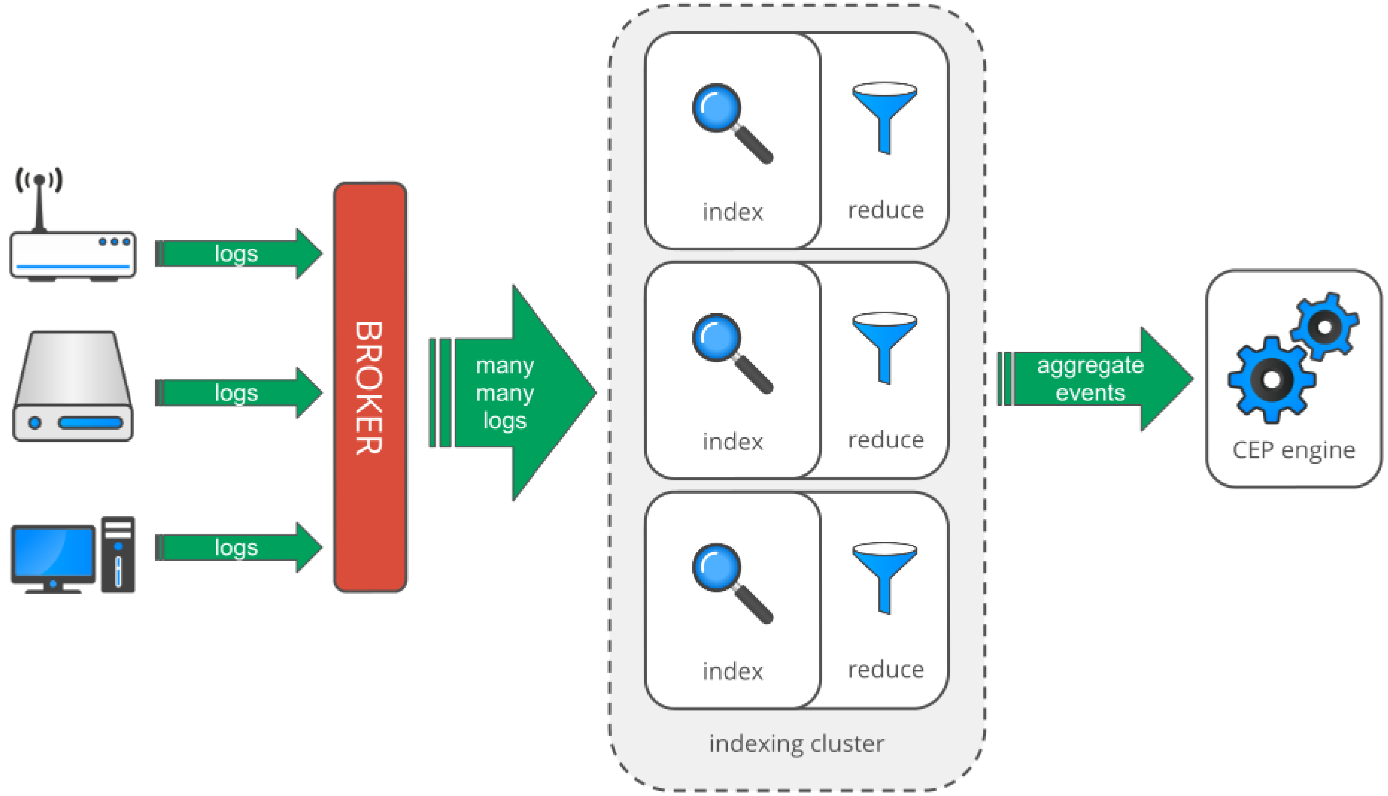
“[...] we’re measuring data less in terms of scale and more in terms of bandwidth.”

Mike Barlow - O'Reilly

Real Time Big Data Analytics: Emerging Architecture

Arquitetura para indexação de Logs

Em produção há cerca de um ano e meio



Alguns números de produção

De Julho de 2013

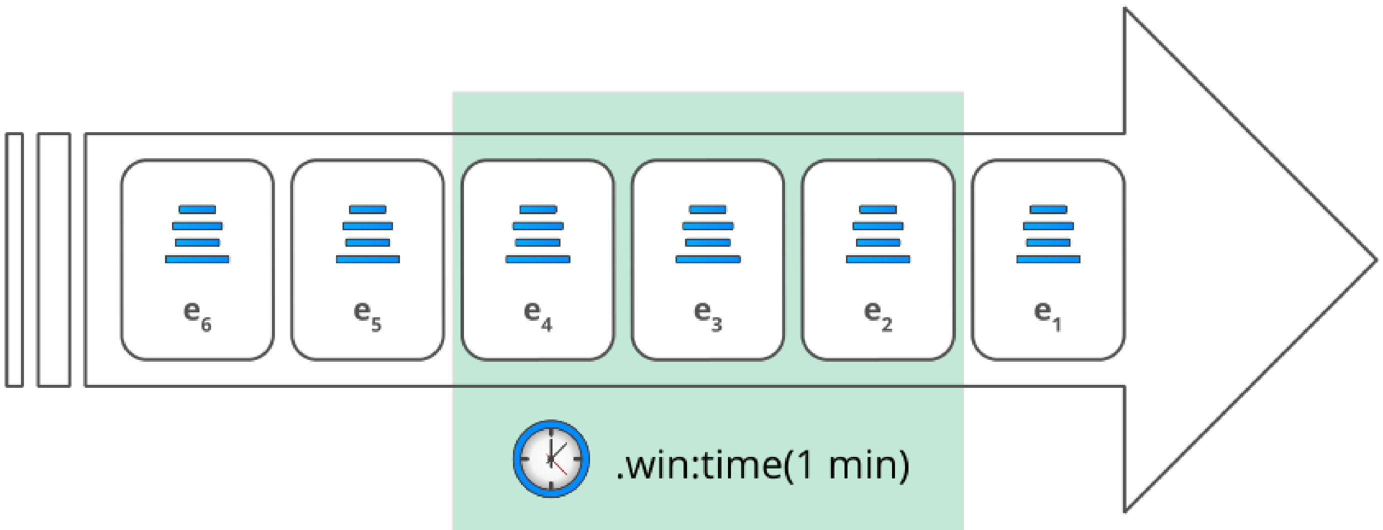
- 4 máquinas físicas no cluster
- Buscas full-text num dataset de 15TB
- Cerca de 60 computações contínuas sobre o fluxo em tempo real
- Média de 20.000 mensagens/segundo (~7 MB/s)
- Máximas de até 80.000 mensagens/segundo em grandes eventos (~30 MB/s)
    - Ao longo de várias horas
    - Exemplo: chegada do Papa no Brasil; final do BBB; jogo do Brasil

Janela de Eventos

Janelas são mantidas em memória

EPL

```
select avg(price) from StockTick(symbol='IBM').win:time(1 min)
```

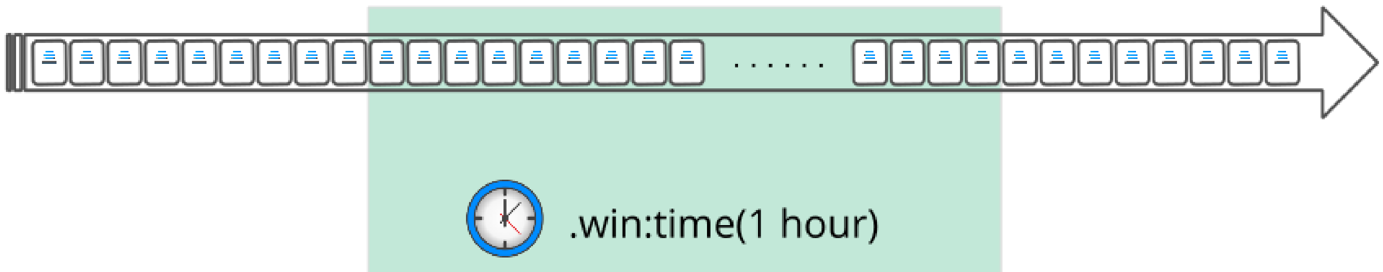


Janela de Eventos

Pode ser ruim com janelas muito longas

EPL

```
select avg(price) from StockTick(symbol='IBM').win:time(1 hour)
```



Intelie Pipes

Linguagem para agregações em tempo real

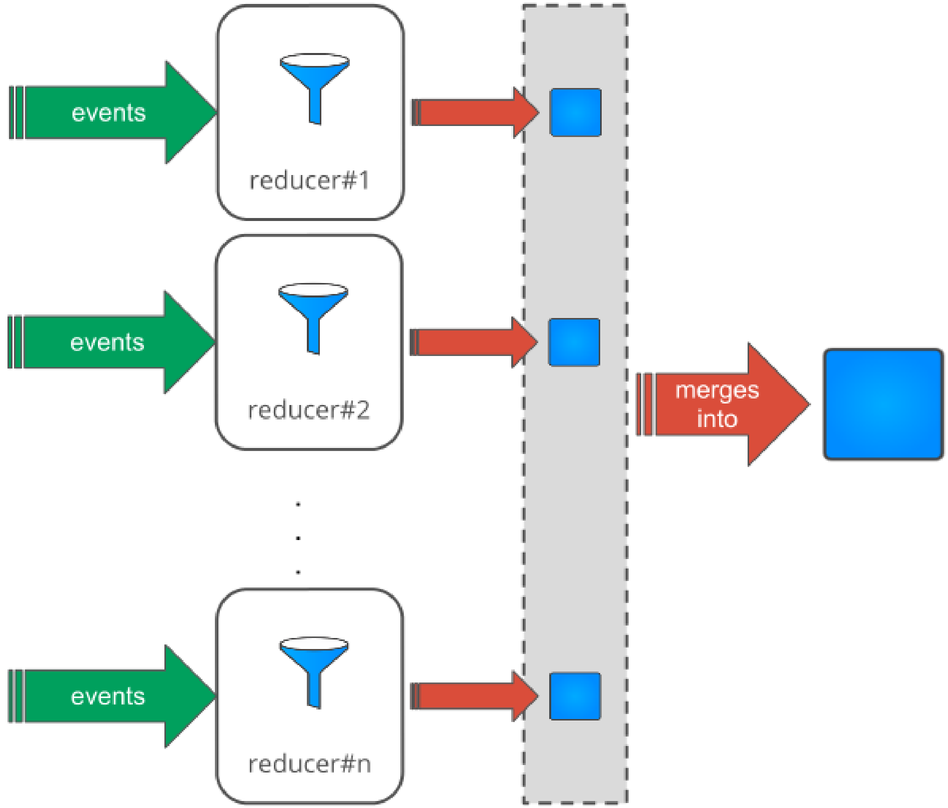
- Agregações sobre grandes fluxos;
- Tolerante a partição;
- O(n) time e O(1) space;
- Associativa à esquerda (onde importa);
- One-liners!!!11

PPL

```
StockTick symbol:IBM => price#avg every minute
```

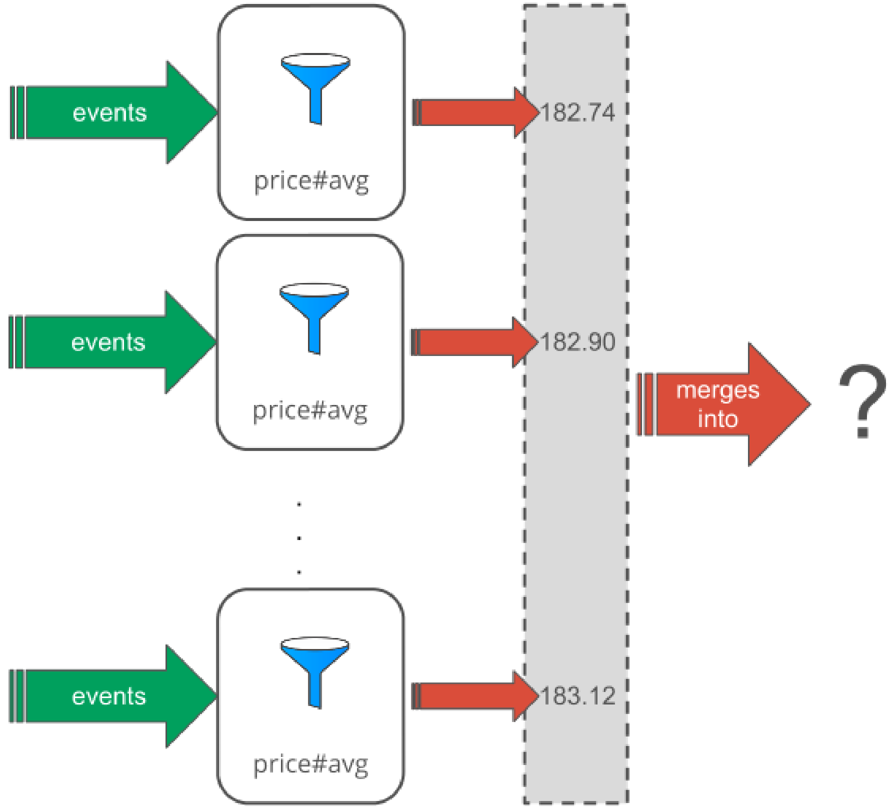
Tolerância a partição

Eu sei que vocês sabem, mas deixa o slide aí



Tolerância a partição

Eu sei que vocês sabem, mas deixa o slide aí



Agregações distribuídas

Mesmo casos simples precisam de cuidado

```
StockTick symbol:IBM => price#stdev every hour
```

PPL

$$\bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

$$M_{2,n} = \sum_{i=1}^{n}(x_i - \bar{x}_n)^2$$

$$M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)$$

$$\sigma_n = \frac{M_{2,n}}{n}$$

$$\delta = \bar{x}_B - \bar{x}_A$$

$$\bar{x}_X = \bar{x}_A + \delta \cdot \frac{n_B}{n_X}$$

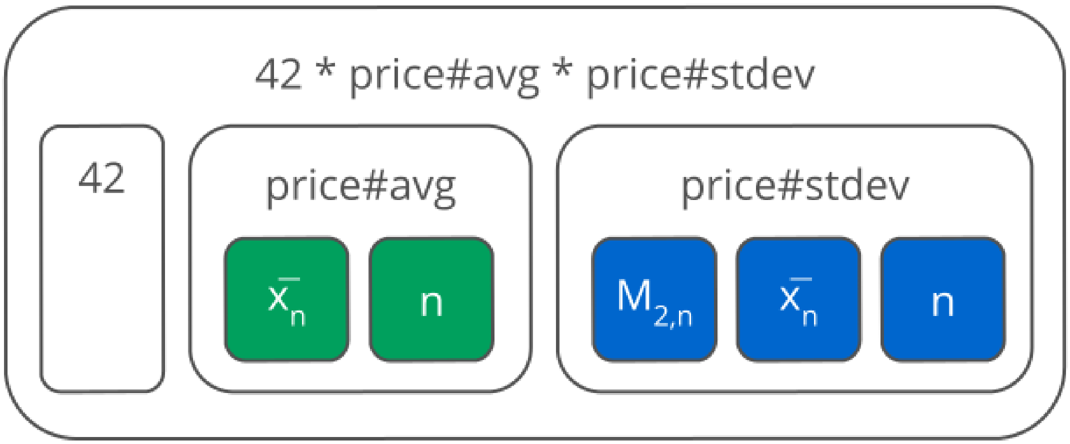
$$M_{2,X} = M_{2,A} + M_{2,B} + \delta^2 \cdot \frac{n_A n_B}{n_X}$$

Agregações distribuídas

Mas afinal, o que é transmitido entre os nós?

PPL

```
StockTick symbol:IBM => 42 * price#avg * price#stdev every hour
```

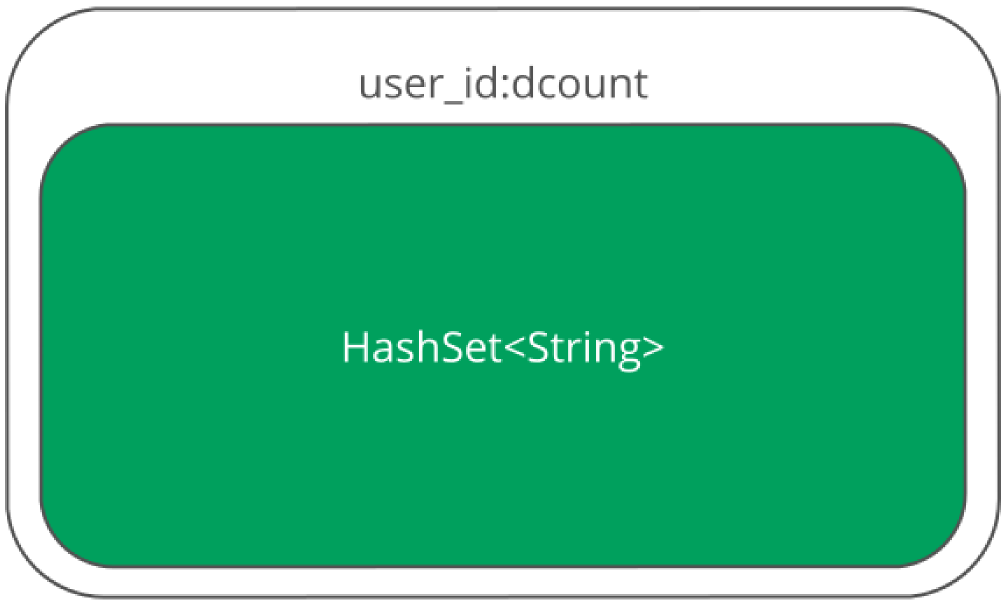


Contar usuários únicos por hora

Moleza, só que não

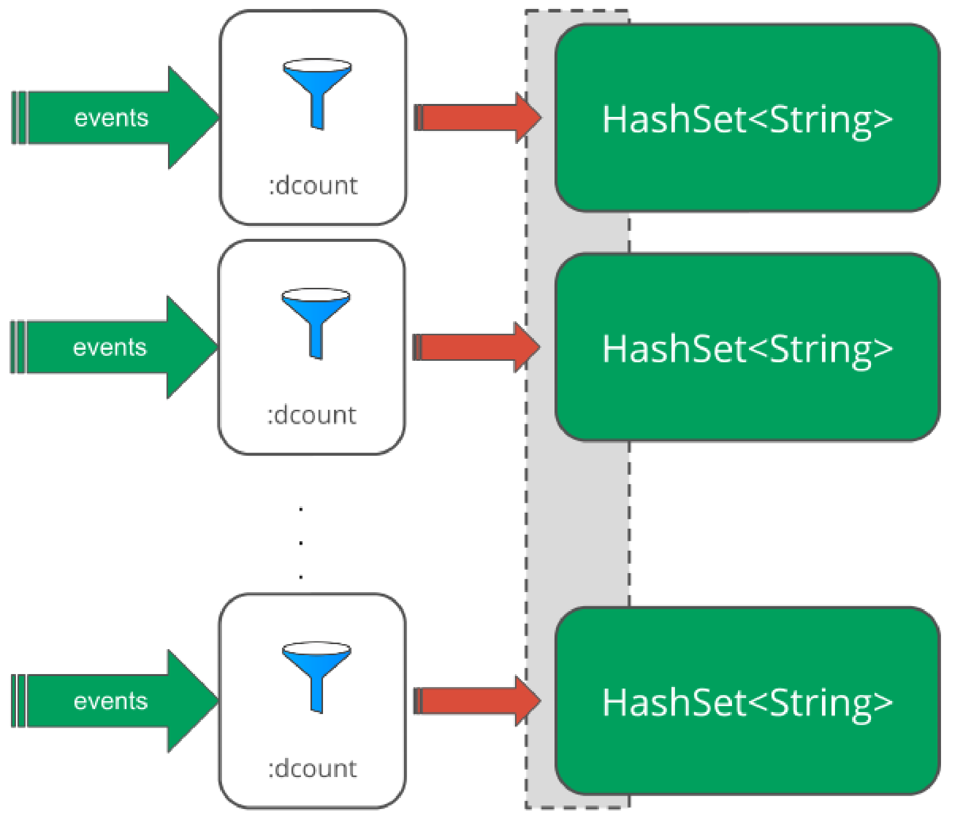
PPL

```
PageViews page:home => user_id:dcount every hour
```



Contar usuários distintos por hora

E o custo de comunicação dos resultados?



Sketching data structures

Porque o universo é probabilístico

(exceto para quem discorda da Interpretação de Copenhagen)

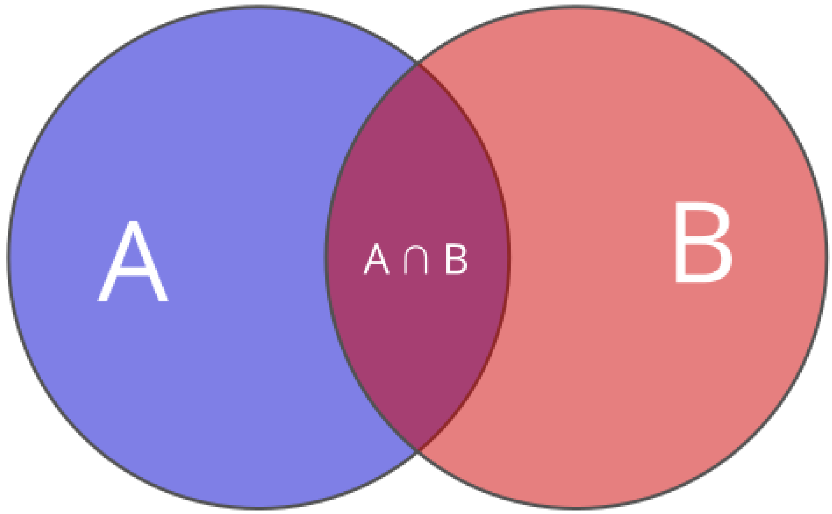
Sketching data structures

Se você pode lidar com um pouco de imprecisão, por que não?

- Apenas uma iteração sobre os dados
  - Úteis para streams (sem acesso aleatório).
- Conhecidos como streaming algorithms
- Espaço e tempo limitados
- Determinismo vs Recursos

Conjuntos

Álgebra: ensinamos isso para crianças de sete anos



$$|A \cap B| = |A| + |B| - |A \cup B|$$

Sketching e conjuntos

Uma área com muitas contribuições recentes.

Set cardinality

HyperLogLog
Philippe Flajolet (2007)

Multiset sumarization

Count-Min Sketch
Graham Cormode (2003)

Set membership

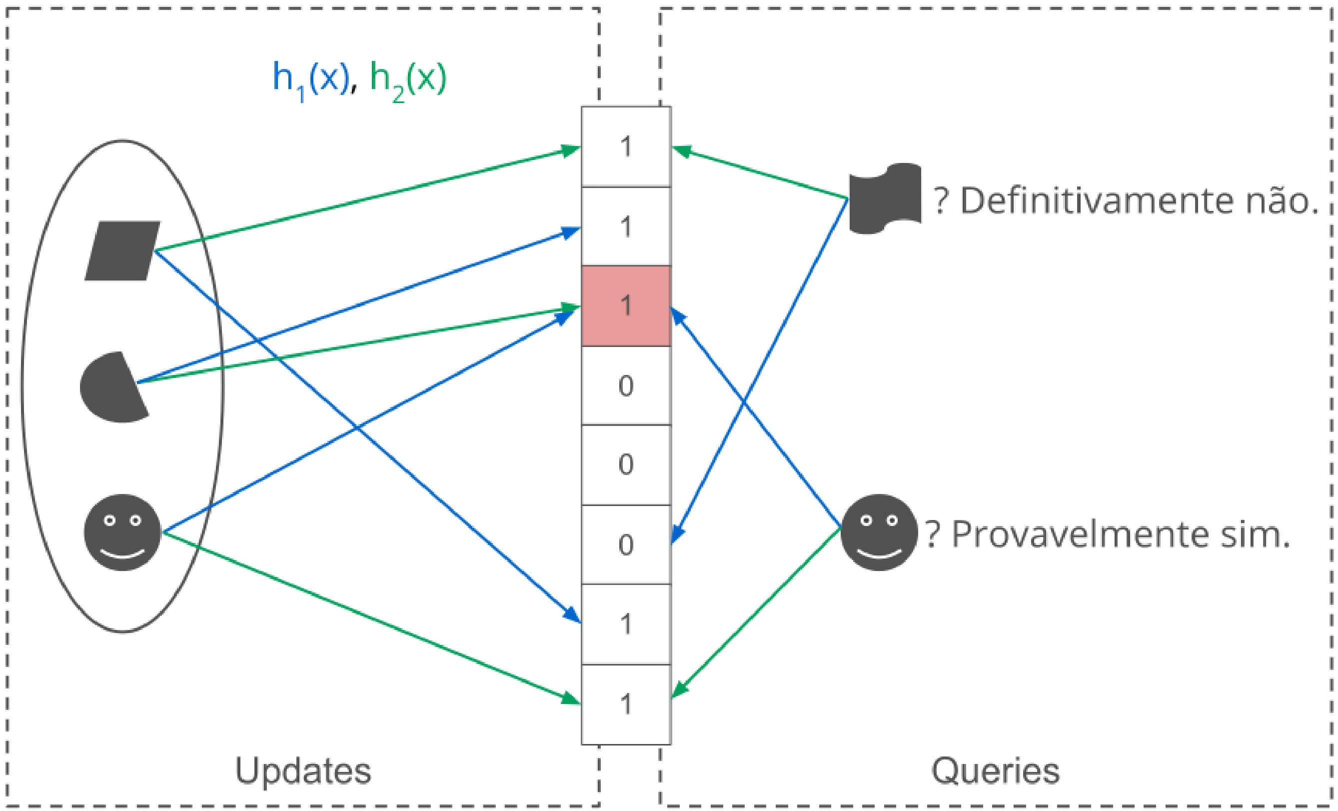
Bloom Filters
Burton Bloom (1970)

“It is theoretically impossible to define a hash function that creates random data from non-random data. But in practice it is not difficult to produce a pretty good imitation.”

Donald Knuth

Bloom Filters

São tipo HashSets, só que não.



Bloom Filters

Probabilidade de falso positivo.

Para n elementos no conjunto, k funções de hash e m bits:

Probabilidade de não-colisão após n inserções com k funções de hash:

$$\left(1 - \frac{1}{m}\right)^{kn}$$

Invertendo, probabilidade de colisão por função de hash:

$$1 - \left(1 - \frac{1}{m}\right)^{kn}$$

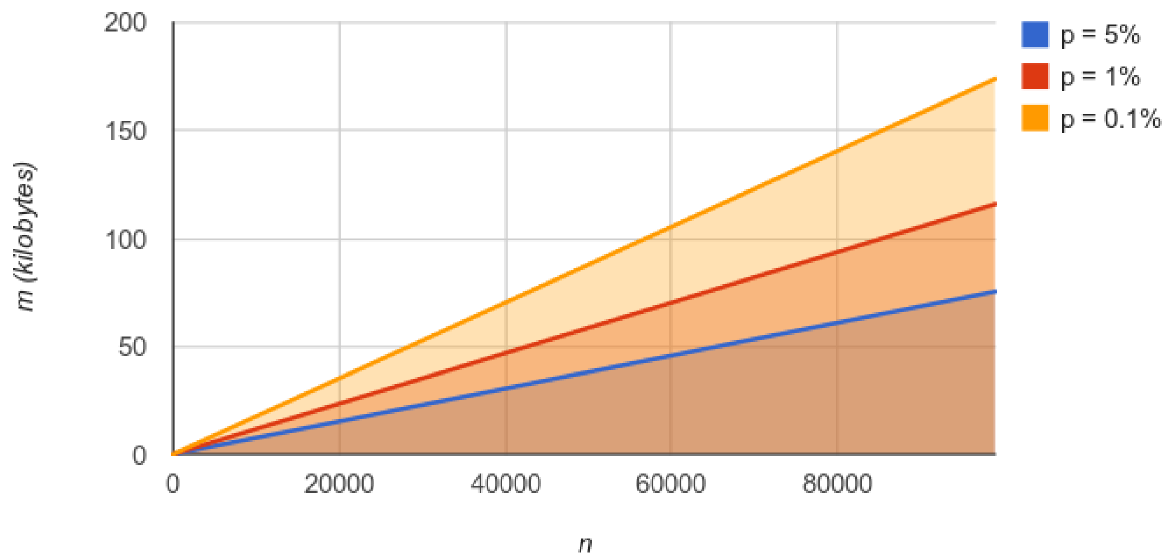
Para k funções de hash:

$$\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^{k}$$

Bloom Filters

Quantos bytes para inserir n itens?

$$m = -\frac{n \ln p}{(\ln 2)^2}$$



“Downloading a few gigabytes of data before you can start using Bitcoin is not a good first user experience. (...) added support for **bloom filters** to get just the transactions relevant to your wallet.”

Gavin Andresen - BitCoin Foundation
https://bitcoinfoundation.org/blog/?p=16

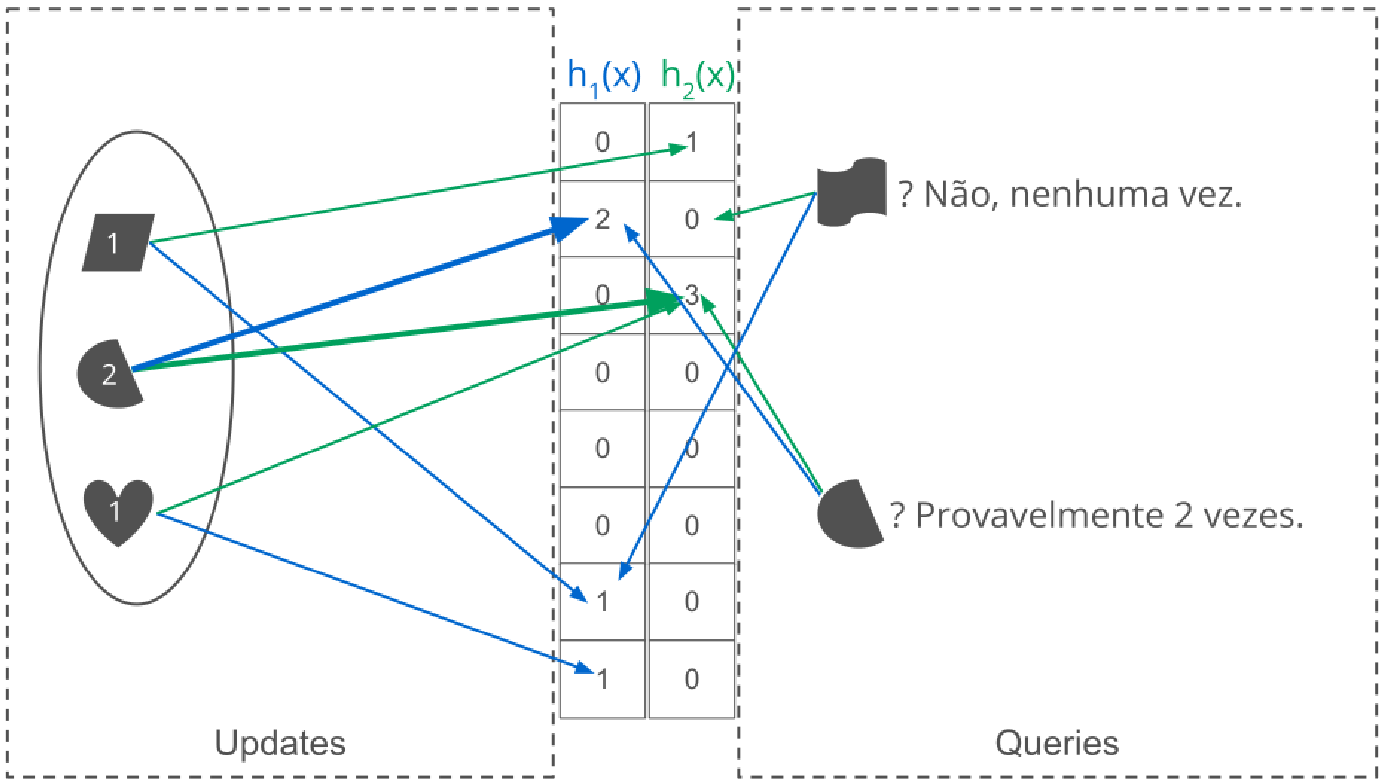
“a small amount of tablet server memory used for storing Bloom Filters drastically reduces the number of disk seeks required for read operations.”

Fay Chang - Google

Bigtable: A Distributed Storage System for Structured Data

Count-Min Sketch

É tipo um Bloom filter, só que não.



Count-Min Sketch

Calculando o erro.

Define-se largura (w) e profundidade (d), de forma que:

$$w = \lceil e/\epsilon \rceil \text{ e } d = \lceil \ln 1/\delta \rceil .$$

Assim, é possível responder point queries ($\hat{a}_i$) onde:

$$a_i \leq \hat{a}_i \leq \epsilon \|a\|$$

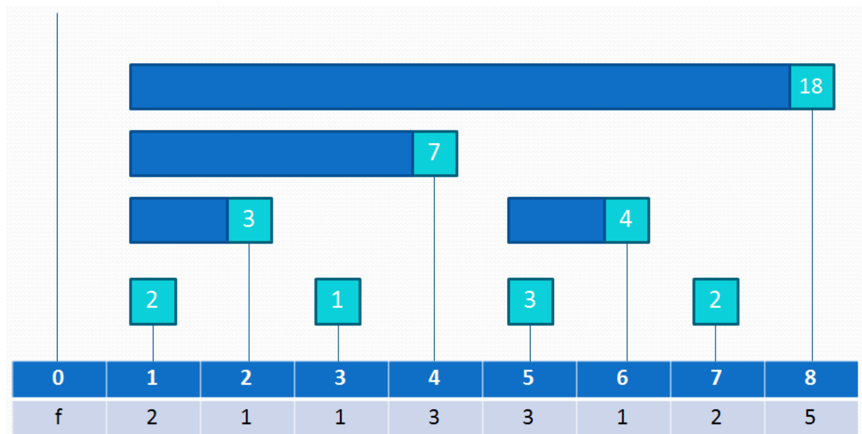
O lado direito da inequação se mantém com probabilidade $1 - \delta$.

Count-Min Sketch

Range queries.

Consiste em estimar $\sum_{i=l}^{r} a_i$.

O truque é tratar a estrutura como uma Fenwick Tree sobre um array probabilistico.



Count-Min Sketch

Uma versão didática (em Python)

PYTHON

```python
class CountMin:
    def __init__(self, m, k):
        self.data = [[0] * m for i in range(k)]
        self.m = m
        self.k = k

    def add(self, element, count=1):
        for i in range(self.k):
            h = mmh3.hash(element, i)
            self.data[i][h % self.m] += count

    def quantile(self, element):
        m = 1<<30
        for i in range(self.k):
            h = mmh3.hash(element, i)
            m = min(m, self.data[i][h % self.m])
        return m
```

“The math establishing error bounds or other properties can be somewhat hairy, but the structures themselves are not too complicated.”

Joseph Kibe - Shareaholic

The Count-min Sketch: How to Count Over Large Keyspaces When 'About Right' Is Good Enough

Phillipe Flajolet (∗1948 †2011)

O homem por trás do HyperLogLog.

HyperLogLog

Distribuição de bits no hash

	0	1	2	3	4	5	6	7
100%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%
	1: 50%	1: 50%	1: 50%	1: 50%	1: 50%	1: 50%	1: 50%	1: 50%
50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%
		1: 50%	1: 50%	1: 50%	1: 50%	1: 50%	1: 50%	1: 50%
25%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%
			1: 50%	1: 50%	1: 50%	1: 50%	1: 50%	1: 50%
12.5%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%	0: 50%
				1: 50%	1: 50%	1: 50%	1: 50%	1: 50%

HyperLogLog

Distribuição de bits no hash

	0	1	2	3	4	5	6	7
1	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2
1/2	0: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2
1/4	0: 1/2	0: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2
1/8	0: 1/2	0: 1/2	0: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2	0: 1/2 1: 1/2

HyperLogLog

Uma ideia genial, estimando por baixo.

Escolhe-se $log_2 m$ bits do espaço de hash para corresponderem a m subfluxos ($|M| = m$). A busca pelo prefixo é efetuada nos bits restantes.

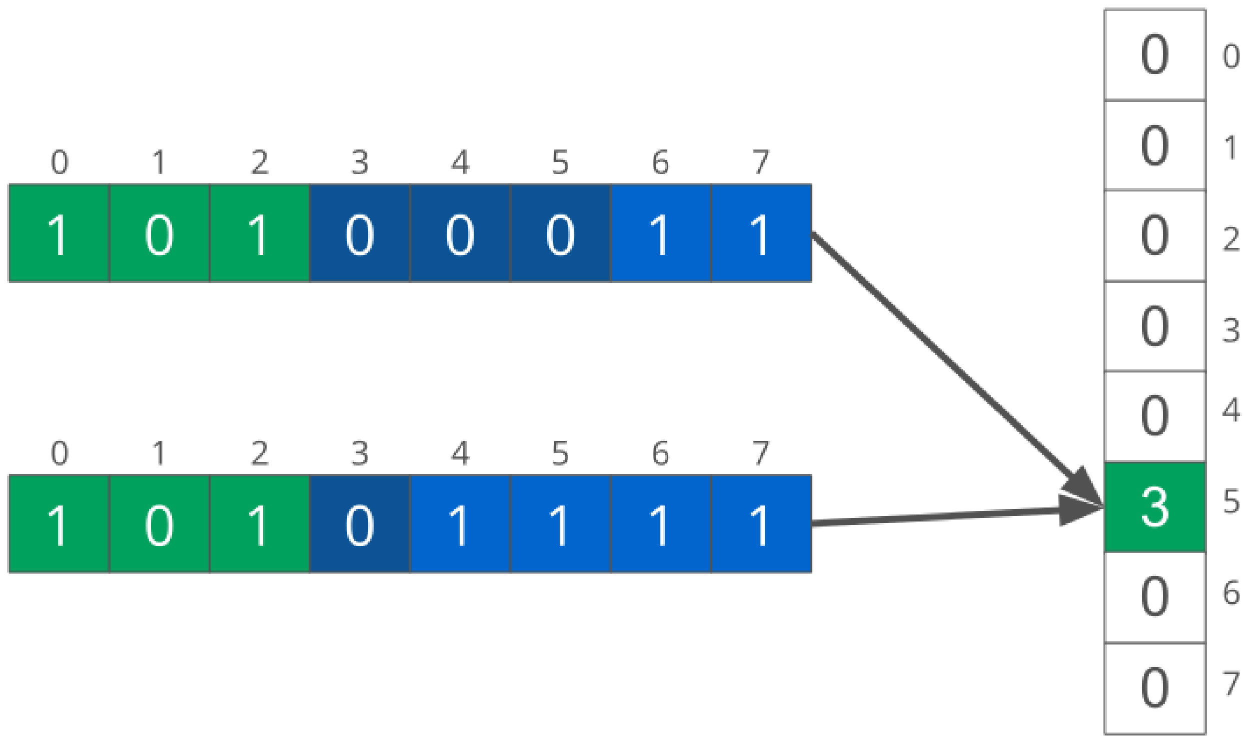
Por exemplo, para $log_2 m = 3$ ($m = 8$):

0	1	2	3	4	5	6	7
1	0	1	0	0	0	1	1

Subfluxo **101** (5)
Valor candidato: **3** (pois há 3 zeros)

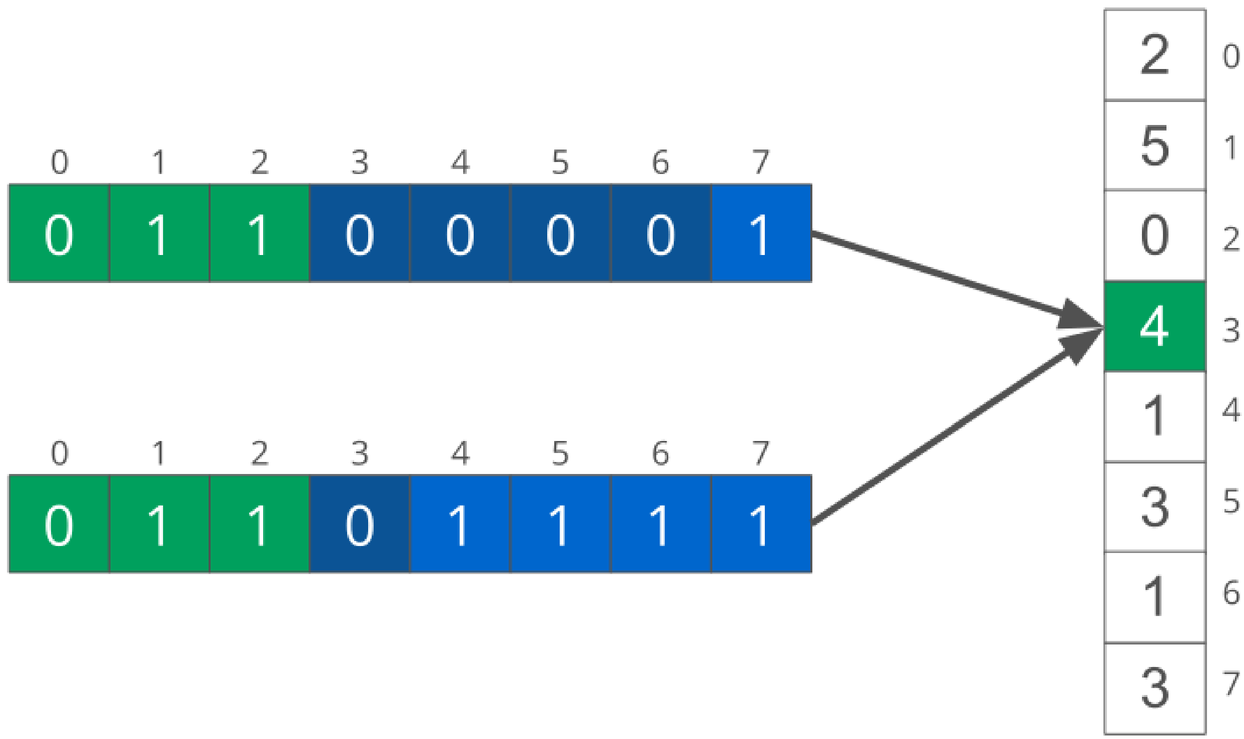
HyperLogLog

Atualizando o array M.



HyperLogLog

Array M depois de diversos updates.



HyperLogLog

Como estimar?

Estimativa de cada registrador:

$$\hat{E}_i = 2^{M(i)}$$

Estimativa geral (média harmônica de todas as estimativas):

$$\hat{E} = \frac{m^2}{\sum_{i=1}^{m} \hat{E}(i)^{-1}} = \frac{m^2}{\sum_{i=1}^{m} 2^{-M(i)}}$$

Estimativa geral (corrigindo o bias multiplicativo):

$$\hat{E}' = \frac{\alpha_m m^2}{\sum_{i=1}^{m} 2^{-M(i)}}, \text{ onde } \alpha_m = (m \int_0^\infty (log_2(\frac{2+u}{1+u}))^m du)^{-1}$$

HyperLogLog

Uma versão didática (em Python, claro)

PYTHON

```python
class HyperLogLog:
    def __init__(self, log2m):
        self.log2m = log2m
        self.m = 1 << log2m
        self.data = [0]*self.m
        self.alphaMM = (0.7213 / (1 + 1.079 / self.m)) * self.m * self.m

    def add(self, element):
        x = mmh3.hash(str(element), 0)
        a, b = 32-self.log2m, self.log2m
        i = x >> a
        v = self._bitscan(x << b, a)
        self.data[i] = max(self.data[i], v)

    def cardinality(self):
        estimate = self.alphaMM / sum([2**-v for v in self.data])
        if estimate <= 2.5 * self.m:
            zeros = float(self.data.count(0))
            return round(-self.m * math.log(zeros / self.m))
        else:
            return round(estimate)
```

Conclusão

Os melhores algoritmos estão nos menores códigos...

... porém nos papers mais obscuros.

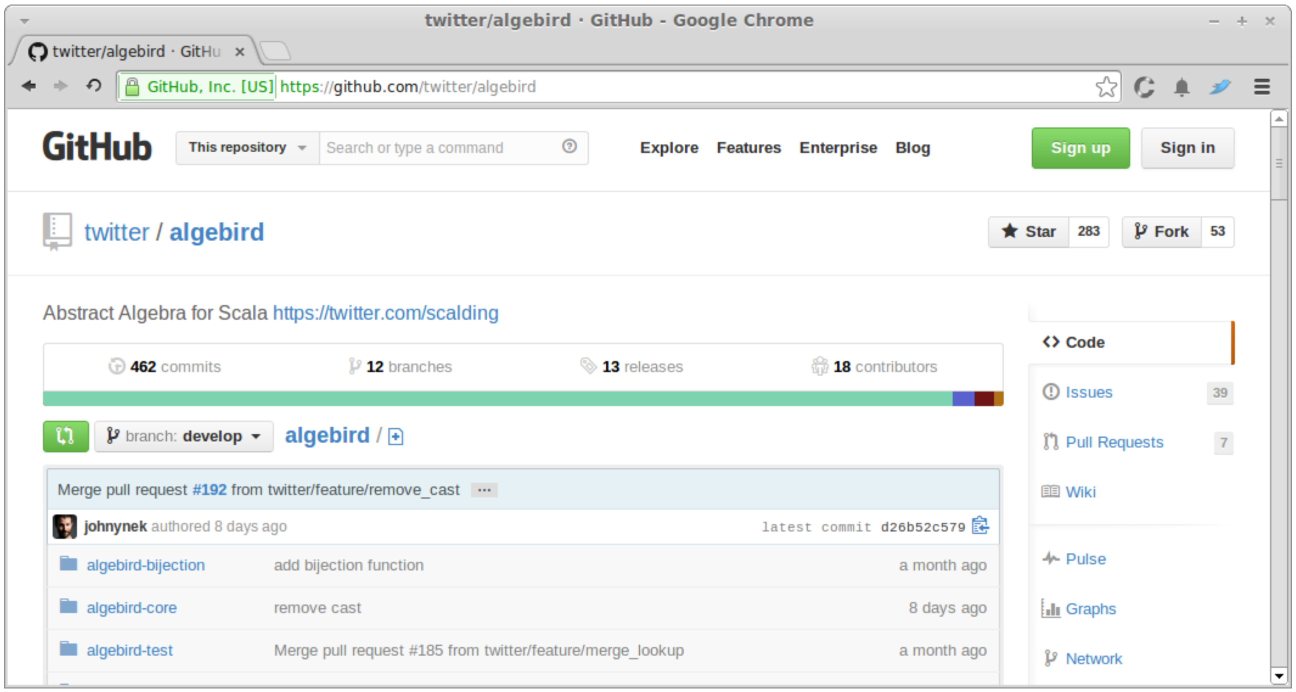
Estruturas de dados probabilisticas vão desempenhar um papel fundamental na evolução do Big Data.

A academia já percebeu isso e já está dez anos à frente.

A indústria ainda está engatinhando.

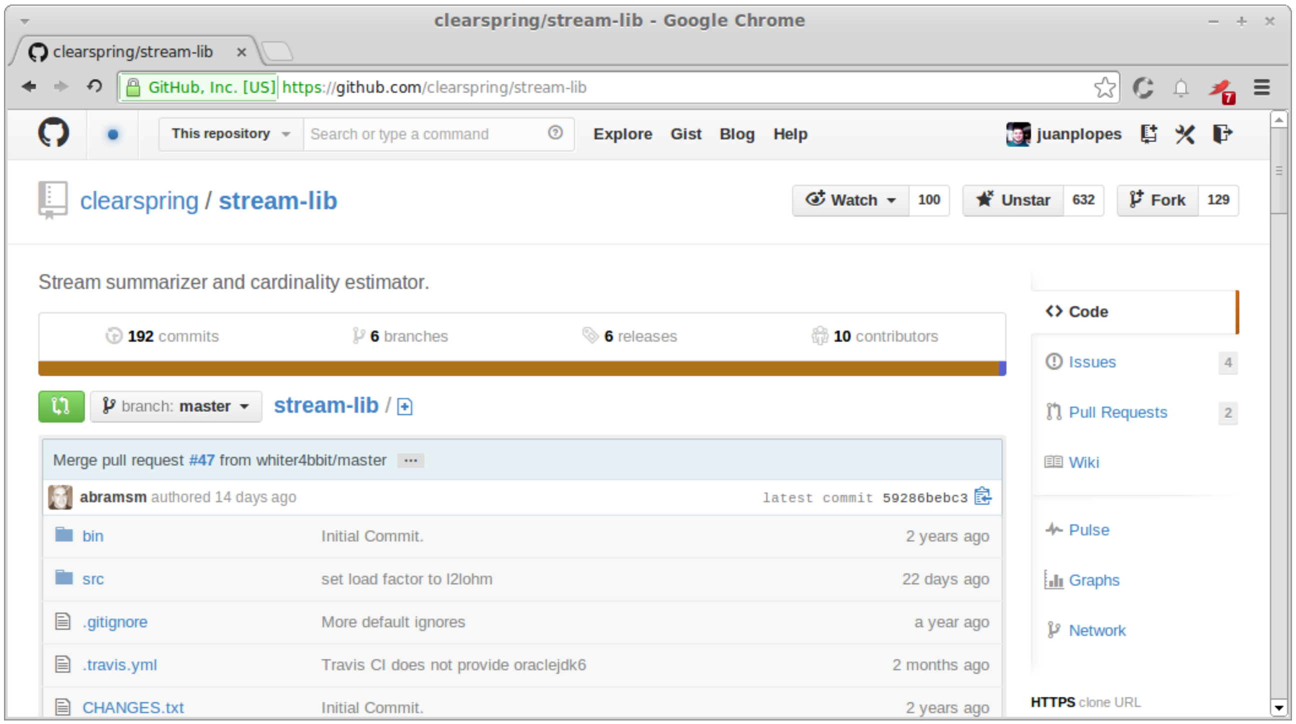
Por falar em indústria...

Twitter Algebird



Por falar em indústria...

ClearSpring (AddThis) stream-lib



Estamos contratando!

intelie.com.br/trabalhe

Dúvidas?

Apresentação em **juanlopes.net/qconsp2013**.

Exemplos em **github.com/juanplopes/sketches**.

Contato:

twitter	@juanplopes
www	juanlopes.net
github	github.com/juanplopes