GERENCIAMENTO DE DADOS EM REDES P2P

Josiane Bezerra (jbf2)
Samuel Arcoverde (sfa)

Roteiro

- Consultas em sistemas P2P
  - Top-k Queries
  - Join Queries
  - Range Queries
- Gerenciamento de Consistência de Réplicas
  - Suporte Básico em DHTs
  - Data Currency em DHTs

Consultas em Sistemas P2P

- Consultas simples, de correspondência exata:
  - Técnicas de encaminhamento
- Buscas mais complexas são difíceis

Consultas Complexas

- Top-k Queries
- Join Queries
- Range Queries

Top-k Queries

- O usuário requisita os k resultados mais relevantes.
- Grau de relevância é determinado por uma função de pontuação (scoring).
- Muito útil em sistemas de gerenciamento de dados em redes P2P, principalmente quando o numero de respostas é muito grande

Top-k Queries

- Consulta:

```
SELECT *
FROM Patient P
WHERE P.disease = ''diabetes''
AND P.height < 170
AND P.weight > 160
ORDER BY scoring-function(height,weight)
STOP AFTER 10
```

Top-k Queries

- A execução eficiente de top-k queries em larga escala em sistemas P2P é difícil
- Técnicas de processamento de top-k query

Algoritmo de Threshold (TA)

- TA é aplicável a consultas onde a função de pontuação é monotônica.

Exemplo

Position	List 1		List 2		List 3	
	Data Item	Local score s_1	Data Item	Local score s_2	Data Item	Local score s_3
1	d_1	30	d_2	28	d_3	30
2	d_4	28	d_6	27	d_5	29
3	d_9	27	d_7	25	d_8	28
4	d_3	26	d_5	24	d_4	25
5	d_7	25	d_9	23	d_2	24
6	d_8	23	d_1	21	d_6	19
7	d_5	17	d_8	20	d_{13}	15
8	d_6	14	d_3	14	d_1	14
9	d_2	11	d_4	13	d_9	12
10	d_{11}	10	d_{14}	12	d_7	11
...	...	...	...	...	...	...

Exemplo

- Scoring function: soma dos scores locais
- Score local de d1 = 30 + 21 + 14 = 65
- Objetivo: Encontrar os k dados com score global mais alto

Algoritmo

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items ;
f: scoring function
Output: Y: list of top-k data items
begin
  $j \leftarrow 1$;
  threshold $\leftarrow 1$;
  min_overall_score $\leftarrow 0$;
  while $j \neq n+1$ *and min_overall_score* $<$ *threshold* **do**
    {Do sorted access in parallel to each of the m sorted lists}
    for *i from* 1 *to m in parallel* **do**
      {Process each data item at position j}
      for *each data item d at position j in* L_i **do**
        {access the local scores of d in the other lists through random access}
        overall_score$(d) \leftarrow f$(scores of d in each L_i)
    $Y \leftarrow k$ data items with highest score so far ;
    min_overall_score $\leftarrow$ smallest overall score of data items in Y ;
    threshold $\leftarrow f$(local scores at position j in each L_i) ;
    $j \leftarrow j+1$
end

Algoritmo

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items ;
f: scoring function
Output: Y: list of top-k data items
begin
  $j \leftarrow 1$;
  threshold $\leftarrow 1$;
  min_overall_score $\leftarrow 0$;
  while $j \neq n+1$ *and min_overall_score* $<$ *threshold* **do**
    {Do sorted access in parallel to each of the m sorted lists}
    for *i from* 1 *to m in parallel* **do**
      {Process each data item at position j}
      for *each data item d at position j in* L_i **do**
        {access the local scores of d in the other lists through random access}
        overall_score$(d) \leftarrow f$(scores of d in each L_i)
    $Y \leftarrow k$ data items with highest score so far ;
    min_overall_score $\leftarrow$ smallest overall score of data items in Y ;
    threshold $\leftarrow f$(local scores at position j in each L_i) ;
    $j \leftarrow j+1$
end

Algoritmo

```
Input: L_1, L_2, ..., L_m: m sorted lists of n data items ;
f: scoring function
Output: Y: list of top-k data items
begin
    j ← 1 ;
    threshold ← 1 ;
    min_overall_score ← 0 ;
    while j ≠ n+1 and min_overall_score < threshold do
        {Do sorted access in parallel to each of the m sorted lists}
        for i from 1 to m  in parallel do
            {Process each data item at position j}
            for each data item d at position j in L_i do
                {access the local scores of d in the other lists through random
                access}
                overall_score(d) ← f(scores of d in each L_i)
        Y ← k data items with highest score so far ;
        min_overall_score ← smallest overall score of data items in Y ;
        threshold ← f(local scores at position j in each L_i) ;
        j ← j+1
end
```

Algoritmo

```
Input: L_1, L_2, ..., L_m: m sorted lists of n data items ;
f: scoring function
Output: Y: list of top-k data items
begin
    j ← 1 ;
    threshold ← 1 ;
    min_overall_score ← 0 ;
    while j ≠ n+1 and min_overall_score < threshold do
        {Do sorted access in parallel to each of the m sorted lists}
        for i from 1 to m  in parallel do
            {Process each data item at position j}
            for each data item d at position j in L_i do
                {access the local scores of d in the other lists through random
                access}
                overall_score(d) ← f(scores of d in each L_i)
        Y ← k data items with highest score so far ;
        min_overall_score ← smallest overall score of data items in Y ;
        threshold ← f(local scores at position j in each L_i) ;
        j ← j+1
end
```

Algoritmo

```
Input: L_1, L_2, ..., L_m: m sorted lists of n data items ;
f: scoring function
Output: Y: list of top-k data items
begin
    j ← 1 ;
    threshold ← 1 ;
    min_overall_score ← 0 ;
    while j ≠ n+1 and min_overall_score < threshold do
        {Do sorted access in parallel to each of the m sorted lists}
        for i from 1 to m  in parallel do
            {Process each data item at position j}
            for each data item d at position j in L_i do
                {access the local scores of d in the other lists through random
                access}
                overall_score(d) ← f(scores of d in each L_i)
        Y ← k data items with highest score so far ;
        min_overall_score ← smallest overall score of data items in Y ;
        threshold ← f(local scores at position j in each L_i) ;
        j ← j+1
end
```

Algoritmo

```
Input: L_1, L_2, ..., L_m: m sorted lists of n data items ;
f: scoring function
Output: Y: list of top-k data items
begin
    j ← 1 ;
    threshold ← 1 ;
    min_overall_score ← 0 ;
    while j ≠ n+1 and min_overall_score < threshold do
        {Do sorted access in parallel to each of the m sorted lists}
        for i from 1 to m  in parallel do
            {Process each data item at position j}
            for each data item d at position j in L_i do
                {access the local scores of d in the other lists through random
                access}
                overall_score(d) ← f(scores of d in each L_i)
        Y ← k data items with highest score so far ;
        min_overall_score ← smallest overall score of data items in Y ;
        threshold ← f(local scores at position j in each L_i) ;
        j ← j+1
end
```

Algoritmo

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items ;
f: scoring function
Output: Y: list of top-k data items
begin
  $j \leftarrow 1$;
  threshold $\leftarrow 1$;
  min_overall_score $\leftarrow 0$;
  while $j \neq n+1$ *and min_overall_score* $<$ *threshold* **do**
    {Do sorted access in parallel to each of the m sorted lists}
    for *i from* 1 *to m in parallel* **do**
      {Process each data item at position j}
      for *each data item d at position j in* L_i **do**
        {access the local scores of d in the other lists through random access}
        overall_score(d) $\leftarrow f$(scores of d in each L_i)
    $Y \leftarrow k$ data items with highest score so far ;
    min_overall_score $\leftarrow$ smallest overall score of data items in Y ;
    threshold $\leftarrow f$(local scores at position j in each L_i) ;
    $j \leftarrow j+1$
end

Exemplo

- Score global de d1: 30+21+14 = 65

Exemplo

Position	List 1		List 2		List 3	
	Data Item	Local score s_1	Data Item	Local score s_2	Data Item	Local score s_3
1	d_1	30	d_2	28	d_3	30
2	d_4	28	d_6	27	d_5	29
3	d_9	27	d_7	25	d_8	28
4	d_3	26	d_5	24	d_4	25
5	d_7	25	d_9	23	d_2	24
6	d_8	23	d_1	21	d_6	19
7	d_5	17	d_8	20	d_{13}	15
8	d_6	14	d_3	14	d_1	14
9	d_2	11	d_4	13	d_9	12
10	d_{11}	10	d_{14}	12	d_7	11
...	...	...	...	...	...	...

Exemplo

- Score global de d1: 30+21+14 = 65
- Score global de d1: 11+28+24 = 63

Exemplo

Position	List 1		List 2		List 3	
	Data Item	Local score s_1	Data Item	Local score s_2	Data Item	Local score s_3
1	d_1	30	d_2	28	d_3	30
2	d_4	28	d_6	27	d_5	29
3	d_9	27	d_7	25	d_8	28
4	d_3	26	d_5	24	d_4	25
5	d_7	25	d_9	23	d_2	24
6	d_8	23	d_1	21	d_6	19
7	d_5	17	d_8	20	d_{13}	15
8	d_6	14	d_3	14	d_1	14
9	d_2	11	d_4	13	d_9	12
10	d_{11}	10	d_{14}	12	d_7	11
...	...	...	...	...	...	...

Exemplo

- Score global de d1: 30+21+14 = 65
- Score global de d1: 11+28+24 = 63
- Score global de d1: 26+14+30 = 70

Exemplo

- 1: Y = {d1, d2, d3}
- 2: Y = {d3, d4, d5}
- 3: Y = {d3, d5, d8}
- ...
- 6: Y = {d3, d5, d8}

Three Phase Uniform Threshold (TPUT)

- Extensão do algoritmo de threshold
- Executa em três fases

Primeira fase

- Os top k dados de cada lista são obtidos
- São escolhidos os k dados com maior score parcial
- A soma parcial do k-ésimo dado é denotada por λ_1.

Segunda Fase

- O *threshold* τ é definido como sendo λ_1/m
- De cada lista são selecionados todos os dados com score local maior ou igual a τ
- A soma parcial do k-ésimo dado é denotada por λ_2
- É calculado o *upper bound score* (u(d))de cada dado em Y
- Para cada dado, se u(d) é menor que λ_2, o dado é removido de Y

Terceira Fase

- Os k dados com maior score são escolhido dentre os elementos de Y

Algoritmo

Algorithm 16.2: Three Phase Uniform Threshold(TPUT)

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items, each at a different list holder;
f: scoring function
Output: Y: list of top-k data items
begin
  {Phase 1}
  for i *from* 1 *to* m *in parallel* **do**
    $Y \leftarrow$ receive top-k data items from L_i holder
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_1 \leftarrow$partial sum of k-th data item in Z ;
  {Phase 2}
  for i *from* 1 *to* m *in parallel* **do**
    send λ_1/m to L_i's holder ;
    $Y \leftarrow$ all data items from L_i's holder whose local scores are not less than λ_1/m
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_2 \leftarrow$ partial sum of k-th data item in Z ;
  $Y \leftarrow Y - \{$data items in Y whose upper bound score is less than $\lambda_2\}$;
  {Phase 3}
  for i *from* 1 *to* m *in parallel* **do**
    send Y to L_i holder ;
    $Z \leftarrow$ data items from L_i's holder that are in both Y and L_i
  $Y \leftarrow k$ data items with highest overall score in Z
end

Algoritmo

Algorithm 16.2: Three Phase Uniform Threshold(TPUT)

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items, each at a different list holder;
f: scoring function
Output: Y: list of top-k data items
begin
  {Phase 1}
  for *i from* 1 *to* m *in parallel* **do**
    $Y \leftarrow$ receive top-k data items from L_i holder
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_1 \leftarrow$ partial sum of k-th data item in Z ;
  {Phase 2}
  for *i from* 1 *to* m *in parallel* **do**
    send λ_1/m to L_i's holder ;
    $Y \leftarrow$ all data items from L_i's holder whose local scores are not less than λ_1/m
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_2 \leftarrow$ partial sum of k-th data item in Z ;
  $Y \leftarrow Y - \{$data items in Y whose upper bound score is less than $\lambda_2\}$;
  {Phase 3}
  for *i from* 1 *to* m *in parallel* **do**
    send Y to L_i holder ;
    $Z \leftarrow$ data items from L_i's holder that are in both Y and L_i
  $Y \leftarrow k$ data items with highest overall score in Z
end

Algoritmo

Algorithm 16.2: Three Phase Uniform Threshold(TPUT)

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items, each at a different list holder;
f: scoring function
Output: Y: list of top-k data items
begin
  {Phase 1}
  for *i from* 1 *to* m *in parallel* **do**
    $Y \leftarrow$ receive top-k data items from L_i holder
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_1 \leftarrow$ partial sum of k-th data item in Z ;
  {Phase 2}
  for *i from* 1 *to* m *in parallel* **do**
    send λ_1/m to L_i's holder ;
    $Y \leftarrow$ all data items from L_i's holder whose local scores are not less than λ_1/m
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_2 \leftarrow$ partial sum of k-th data item in Z ;
  $Y \leftarrow Y - \{$data items in Y whose upper bound score is less than $\lambda_2\}$;
  {Phase 3}
  for *i from* 1 *to* m *in parallel* **do**
    send Y to L_i holder ;
    $Z \leftarrow$ data items from L_i's holder that are in both Y and L_i
  $Y \leftarrow k$ data items with highest overall score in Z
end

Algoritmo

Algorithm 16.2: Three Phase Uniform Threshold(TPUT)

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items, each at a different list holder;
f: scoring function
Output: Y: list of top-k data items
begin
  {Phase 1}
  for *i from* 1 *to* m *in parallel* **do**
    $Y \leftarrow$ receive top-k data items from L_i holder
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_1 \leftarrow$ partial sum of k-th data item in Z ;
  {Phase 2}
  for *i from* 1 *to* m *in parallel* **do**
    send λ_1/m to L_i's holder ;
    $Y \leftarrow$ all data items from L_i's holder whose local scores are not less than λ_1/m
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_2 \leftarrow$ partial sum of k-th data item in Z ;
  $Y \leftarrow Y - \{$data items in Y whose upper bound score is less than $\lambda_2\}$;
  {Phase 3}
  for *i from* 1 *to* m *in parallel* **do**
    send Y to L_i holder ;
    $Z \leftarrow$ data items from L_i's holder that are in both Y and L_i
  $Y \leftarrow k$ data items with highest overall score in Z
end

Algoritmo

Algorithm 16.2: Three Phase Uniform Threshold(TPUT)

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items, each at a different list holder;
f: scoring function
Output: Y: list of top-k data items
begin
  {Phase 1}
  for *i from* 1 *to* m *in parallel* **do**
    $Y \leftarrow$ receive top-k data items from L_i holder
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_1 \leftarrow$ partial sum of k-th data item in Z ;
  {Phase 2}
  for *i from* 1 *to* m *in parallel* **do**
    send λ_1/m to L_i's holder ;
    $Y \leftarrow$ all data items from L_i's holder whose local scores are not less than λ_1/m
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_2 \leftarrow$ partial sum of k-th data item in Z ;
  $Y \leftarrow Y - \{$data items in Y whose upper bound score is less than $\lambda_2\}$;
  {Phase 3}
  for *i from* 1 *to* m *in parallel* **do**
    send Y to L_i holder ;
    $Z \leftarrow$ data items from L_i's holder that are in both Y and L_i
  $Y \leftarrow k$ data items with highest overall score in Z
end

Algoritmo

Algorithm 16.2: Three Phase Uniform Threshold(TPUT)

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items, each at a different list holder;
f: scoring function
Output: Y: list of top-k data items
begin
  {Phase 1}
  for *i from* 1 *to* m *in parallel* **do**
    $Y \leftarrow$ receive top-k data items from L_i holder
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_1 \leftarrow$ partial sum of k-th data item in Z ;
  {Phase 2}
  for *i from* 1 *to* m *in parallel* **do**
    send λ_1/m to L_i's holder ;
    $Y \leftarrow$ all data items from L_i's holder whose local scores are not less than λ_1/m
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_2 \leftarrow$ partial sum of k-th data item in Z ;
  $Y \leftarrow Y - \{$data items in Y whose upper bound score is less than $\lambda_2\}$;
  {Phase 3}
  for *i from* 1 *to* m *in parallel* **do**
    send Y to L_i holder ;
    $Z \leftarrow$ data items from L_i's holder that are in both Y and L_i
  $Y \leftarrow k$ data items with highest overall score in Z
end

Algoritmo

Algorithm 16.2: Three Phase Uniform Threshold(TPUT)

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items, each at a different list holder;
f: scoring function
Output: Y: list of top-k data items
begin
  {Phase 1}
  for *i from* 1 *to* m *in parallel* **do**
    $Y \leftarrow$ receive top-k data items from L_i holder
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_1 \leftarrow$ partial sum of k-th data item in Z ;
  {Phase 2}
  for *i from* 1 *to* m *in parallel* **do**
    send λ_1/m to L_i's holder ;
    $Y \leftarrow$ all data items from L_i's holder whose local scores are not less than λ_1/m
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_2 \leftarrow$ partial sum of k-th data item in Z ;
  $Y \leftarrow Y - \{$data items in Y whose upper bound score is less than $\lambda_2\}$;
  {Phase 3}
  for *i from* 1 *to* m *in parallel* **do**
    send Y to L_i holder ;
    $Z \leftarrow$ data items from L_i's holder that are in both Y and L_i
  $Y \leftarrow k$ data items with highest overall score in Z
end

Algoritmo

Algorithm 16.2: Three Phase Uniform Threshold(TPUT)

Input: $L_1, L_2, \ldots, L_m$: m sorted lists of n data items, each at a different list holder;
f: scoring function
Output: Y: list of top-k data items
begin
  {Phase 1}
  for *i from* 1 *to* m *in parallel* **do**
    $Y \leftarrow$ receive top-k data items from L_i holder
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_1 \leftarrow$ partial sum of k-th data item in Z ;
  {Phase 2}
  for *i from* 1 *to* m *in parallel* **do**
    send λ_1/m to L_i's holder ;
    $Y \leftarrow$ all data items from L_i's holder whose local scores are not less than λ_1/m
  $Z \leftarrow$ data items with the k highest partial sum in Y ;
  $\lambda_2 \leftarrow$ partial sum of k-th data item in Z ;
  $Y \leftarrow Y - \{$data items in Y whose upper bound score is less than $\lambda_2\}$;
  {Phase 3}
  for *i from* 1 *to* m *in parallel* **do**
    send Y to L_i holder ;
    $Z \leftarrow$ data items from L_i's holder that are in both Y and L_i
  $Y \leftarrow k$ data items with highest overall score in Z
end

Exemplo

Position	List 1		List 2	
	Data Item	Local score s_1	Data Item	Local score s_2
1	d_1	30	d_2	28
2	d_4	28	d_6	27
3	d_9	27	d_7	25
4	d_3	26	d_5	24
5	d_7	25	d_9	23
6	d_8	23	d_1	21
7	d_5	17	d_8	20
8	d_6	14	d_3	14
9	d_2	11	d_4	13
10	d_{11}	10	d_{14}	12
...	...	...	...	...

Exemplo

- Primeira fase
  - Y = {d1;d2;d4;d6}
  - Z = {d1;d2}
  - $\lambda_1 = 28$
- Segunda fase
  - $\tau = 14$
  - Y = {(d_1;30;21); (d_2;0;28); (d_3;26;14); (d_4;28;0); (d_5;17;24); (d_6;14;27); (d_7;25;25); (d_8;23;20); (d_9;27;23)}
  - Z = {(d_1;30;21); (d_7;25;25)}

Exemplo

- *Upper bound scores:*
  - $u(d_1) = 30+21 = 51$
  - $u(d_2) = 14+28 = 42$
  - $u(d_3) = 26+14 = 40$
  - $u(d_4) = 28+14 = 42$
  - $u(d_5) = 17+24 = 41$
  - $u(d_6) = 14+27 = 41$
  - $u(d_7) = 25+25 = 50$
  - $u(d_8) = 23+20 = 43$
  - $u(d_9) = 27+23 = 50$
- *Y = {d1, d7, d9}*

Exemplo

- Terceira fase:
  - Y = {d1, d7} ou {d1, d9}

Best Position Algorithm (BPA)

- Executa top-k queries de forma mais eficiente que o TA
- A condição de parada se baseia no score global, calculado com base nas melhores posições em todas as listas
- A melhor posição em uma lista é a mais alta posição tal que qualquer posição antes dela também foi acessada.

Best Position Algorithm (BPA)

- Seja P_i o conjunto de posições as quais foram acessadas randomica ou sequencialemte em L_i
- Seja bp_i a melhor posição em L_i: mais alto valor em P_i, de forma que, para qualquer posição de L_i, entre 1 e bp_i, também está em P_i.
- E seja $s_i(bp_i)$ o score local do dado na posição bp_i em L_i
- Entao, o threshold é $f(s1(bp1); s2(bp2); ...; sm(bpm))$ para alguma função f.

Exemplo

Position	List 1		List 2		List 3	
	Data Item	Local score s_1	Data Item	Local score s_2	Data Item	Local score s_3
1	d_1	30	d_2	28	d_3	30
2	d_4	28	d_6	27	d_5	29
3	d_9	27	d_7	25	d_8	28
4	d_3	26	d_5	24	d_4	25
5	d_7	25	d_9	23	d_2	24
6	d_8	23	d_1	21	d_6	19
7	d_5	17	d_8	20	d_{13}	15
8	d_6	14	d_3	14	d_1	14
9	d_2	11	d_4	13	d_9	12
10	d_{11}	10	d_{14}	12	d_7	11
...	...	...	...	...	...	...

Exemplo

- Posição 1:
  - P1 = {1, 4, 9}
  - $bp_1 = 1$
  - P2 = {1,6,8}
  - $bp_2 = 1$
  - P3 = {1, 5, 8}
  - $bp_3 = 1$
  - Score global da melhor posição: $\lambda = f(s_1(1), s_2(1), s_3(1)) = 88$
  - Socre global dos dados em P1 é menor que λ, entao o algoritmo não para.

Exemplo

- Posição 2:
  - P1 = {1, 2, 4, 7, 8, 9}
  - $bp_1 = 2$
  - P2 = {1,2, 4, 6, 8, 9}
  - $bp_2 = 2$
  - P3 = {1, 2, 4, 5, 6, 8}
  - $bp_3 = 2$
  - Score global da melhor posição: $\lambda = f(s_1(2), s_2(2), s_3(2)) = 84$
  - Socre global dos dados em P2 é menor que λ, então o algoritmo não para.

Exemplo

- Posição 3:
  - P1 = {1, 2, 3, 4, 5, 6, 7, 8, 9}
  - $bp_1 = 9$
  - P2 = {1, 2, 3, 4, 5, 6, 7, 8, 9}
  - $bp_2 = 9$
  - P3 = {1, 2, 3, 4, 5, 6, 7, 8, 10}
  - $bp_3 = 8$
  - Score global da melhor posição: $\lambda = f(s_1(9), s_2(9), s_3(8)) = 38$
  - Socre global de todos os dados em P3 é maior que λ, então o algoritmo não para.

Join Queries

- Os principais algoritmos de junção em BDD utilizam tabelas hash
  - DHTs podem explorá-los nativamente
- Solução básica apresentada pelo PIERjoin

47

PIERjoin

- Assume que as relações de junção são armazenadas a partir de uma fragmentação horizontal
- Cada par só armazena tuplas baseadas nos valores dos atributos da sua fragmentação
- Dada uma consulta, ela é propagada por todos os pares que armazenam tuplas das relações
- Cada par verifica as tuplas que satisfazem o predicado selecionado e envia-os de volta

48

PIERjoin

- Para cada junção da relação é definida uma tabela de hash
- Cada tupla recebida é adicionada na tabela hash da sua relação
- As junções são realizadas com base nos valores recuperados
- Não dá nenhuma garantia sobre a completude do resultado (indisponibilidade)

49

Range Queries

- Consultas que recuperam todos os valores localizados entre um valor inferior e um superior
  - BETWEEN 3 and 7

50

Range Queries

- Sistemas P2P estruturados conseguem realizar consultas de valor exato eficientemente
  - “valor A” = “valor B”
- Não são tão eficientes para consultas de intervalo
  - hashing desordena facilmente os índices
- Duas abordagens:
  - Proximidade
  - Prefixed Hash Tree (PHT)

51

Proximidade

- Direciona, com alta probabilidade, hashs de conteúdo com intervalos parecidos para os mesmos nós DHT
- Obtém apenas respostas aproximadas
- Pode desbalancear as cargas dos nós em grandes redes
- Skipnet
  - DHT que preserva a ordem lexicográfica mantendo itens com valores similares em nós contínuos
  - Cada nó é responsável por um conjunto de strings
- Consulta é simples, mas é realizada em tempo linear

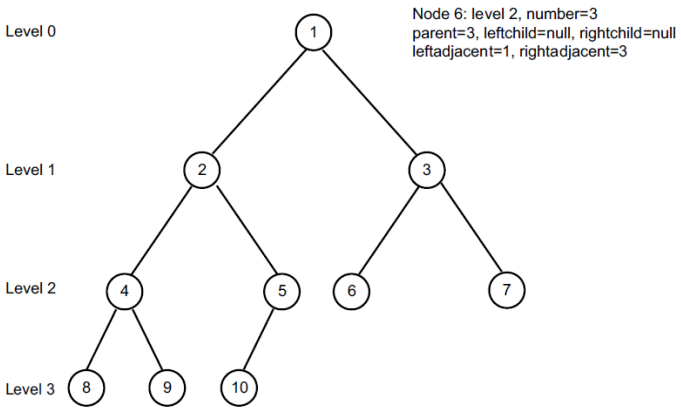
52

Prefix Hash Tree

- Distribuição estruturada dos dados baseadas numa árvore prefixada
- Consulta pode ser realiza por busca binária
- BATON (BAlanced Tree Overlay Network)
  - O primeiro nível começa com o nó raiz
  - Cada nó armazena um link para seu pai, filhos, e nós de mesmo nível

53

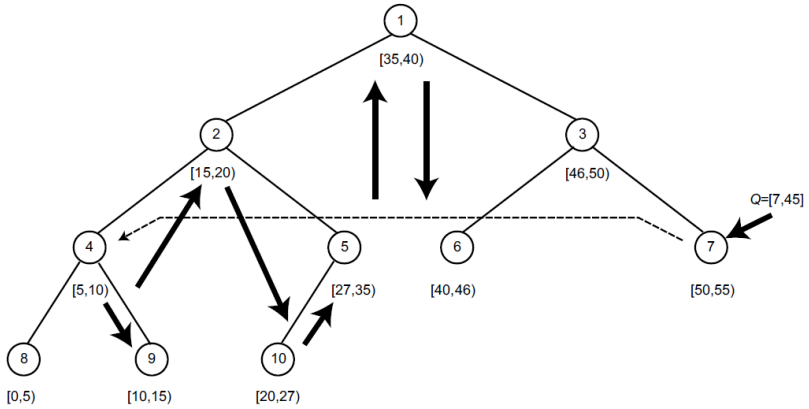
PHT - BATON



54

PHT - BATON

- Consulta realizada em *O(log n)*



55

Top-k queries - sistemas não-estruturados

- Enviar a consulta para todos os nós e recuperar todas as respostas disponíveis
  - Altamente inviável
- Fully Decentralized Top-k (FD)
  - Utiliza TTL
  - Independe da disponibilidade de todos os nós

56

Fully Decentralized Top-k

- A consulta é enviada para os pares acessíveis que estão a uma distância menor que o TTL
- Cada par realiza a consulta localmente e utilizando uma função de ordenação gera uma lista com os top-k
- Os nós esperam os resultados dos seus vizinhos (filhos) antes de enviar a resposta
- Os resultados locais dos vizinhos são mesclados e a referência para os valores é enviada para um nível acima

57

Fully Decentralized Top-k

- Se os nós filhos não enviarem os resultados no tempo estipulado o par atual não precisa mais esperar
- Por fim os dados chegam ao requerente da consulta na forma de uma lista de *scores,* podendo realizar a consulta top-k

58

Top-k queries – sistemas DHT

- Solução faz parte do projeto APPA e baseia-se no TA
- Armazena uma tabela de índices secundários
  - Armazena o valor dos atributos
- Valores de atributos pertencentes ao mesmo subdomínio são armazenados próximos ou no mesmo nó

59

Top-k queries - sistemas super peer

- Algoritmo Edutella
  - Considera que há uma pequena porcentagem de super peers
- O par que gera a consulta envia-a para o seu super peer
- A consulta é enviada para todos os outros super peers
  - Cada um gerencia a quantidade de consultas que vai retornar
- O processo de ordenação é realizado iterativamente
  - Basta apenas saber qual é a melhor resposta
  - Quem a melhor resposta indica a segunda melhor, e assim por diante

60

Consistência de replicação de dados

- Sistemas P2P replicam dados a fim de aumentar:
  - disponibilidade de dados
  - desempenho de acesso
- Sistemas P2P diferentes proveem tipos diferentes de replicação consistente

61

Consistência de replicação de dados

- Inicialmente, os sistemas baseavam-se apenas em dados estáticos
  - Arquivos de música
- Replicação “passiva”
  - Um par faz a requisição de um arquivo e copia-o a partir de outro par
- Sistemas P2P mais avançados utilizam técnicas de gerenciamento de réplicas
  - Réplicas podem ser atualizadas

62

Consistência de replicação de dados

- Abordagens
  - Basic Support in DHTs
  - Data currency in DHTs

63

Basic Support in DHTs

- A maioria dos DHTs baseia-se na replicação com armazenamento de pares (*key, data*) em diversos pares
- Caso um par esteja indisponível, os dados podem ser recuperados por outros pares que mantenham uma réplica
- Duas importantes técnicas:
  - CAN
  - Tapestry

64

CAN

- Assume dados apenas de leitura
- Duas abordagens

65

CAN – 1ª

- Funções hash mapeiam uma chave simples
- Replicação da tupla (*key, data*) para os nós da rede

66

CAN – 2ª

- Melhoramento da primeira
- Um nó replica chaves “populares” nos seus vizinhos
- Chaves replicadas tem um TTL associado

67

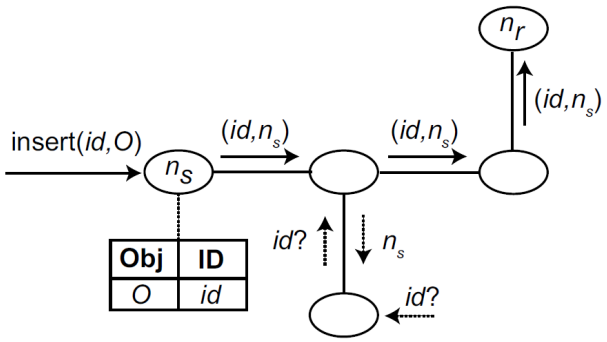
Tapestry

- Localização e encaminhamento descentralizado de objetos
- Roteia mensagens para terminais lógicos
- Dado que um objeto ***O*** seja um objeto identificado por *id (**o**)*, a inserção de ***o*** na rede P2P envolve dois nós:
  - nó servidor (n_s) que mantém ***o***
  - nó raiz (n_r) que mantém o mapeamento na forma (id(o), n_s) indicando que o objeto identificado por *id(o)* está armazenado no nó n_s.

68

Tapestry

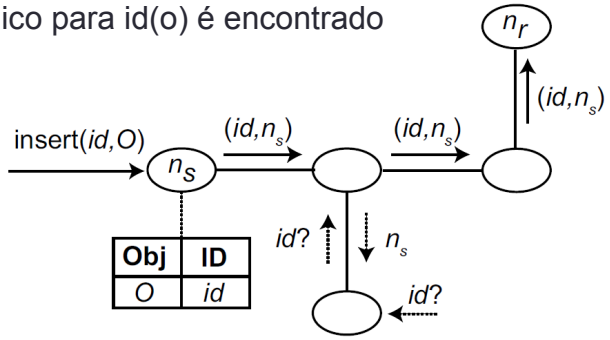
- Nó raiz é dinamicamente determinado por um algoritmo determinístico.
- O mapeamento é armazenado em todos os nós ao longo do caminho percorrido pela mensagem.



69

Tapestry - Consulta

- Busca pelo *id(o)* é inicialmente roteada para n_r
- Ele pode ser encontrado baseado na encontro da chave (id(o), n_s)
- Cada nó encaminha a mensagem para seu vizinho onde o identificador lógico para id(o) é encontrado



70

Tapestry - Gerenciamento de réplicas

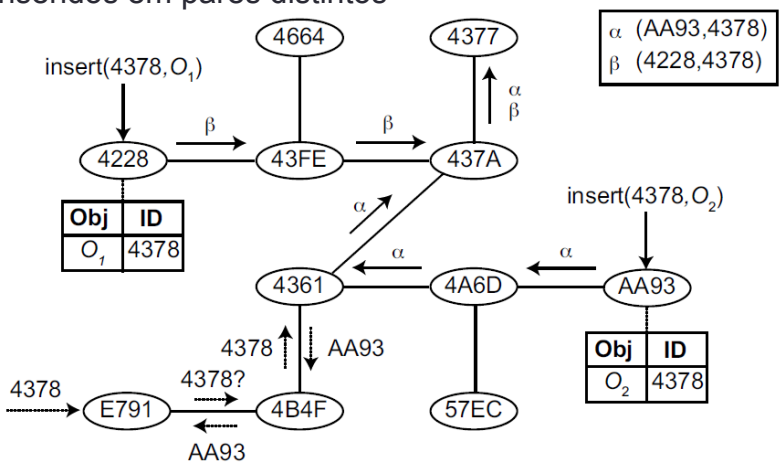
- Cada nó no grafo representa um par na rede P2P
- Identificador lógico de par com formato hexadecimal

α	(AA93,4378)
β	(4228,4378)

71

Tapestry - Gerenciamento de réplicas

- Neste exemplo, duas réplicas O_1 e O_2 do objeto O são inseridos em pares distintos



72

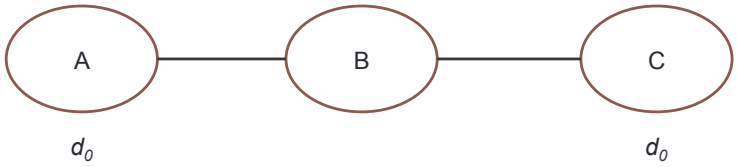
Data Currency em DHTs

- A consistência de réplicas que sofrem atualizações pode ser comprometida
  - Indisponibilidade de pares
  - Réplicas simultâneas

73

Data Currency em DHTs

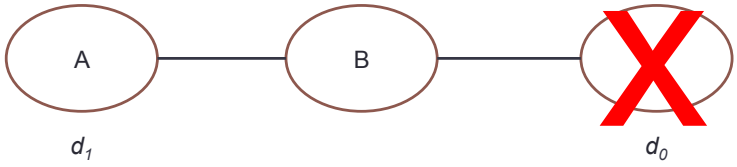
- *put(k, d_0)*



74

Data Currency em DHTs - Problema 1

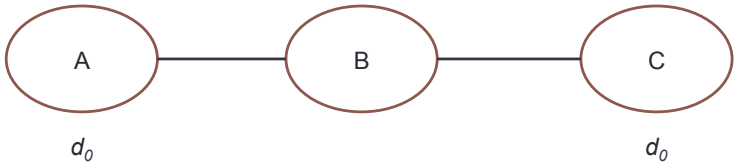
- Indisponibilidade de um par
- *put(k, d_1)*



75

Data Currency em DHTs

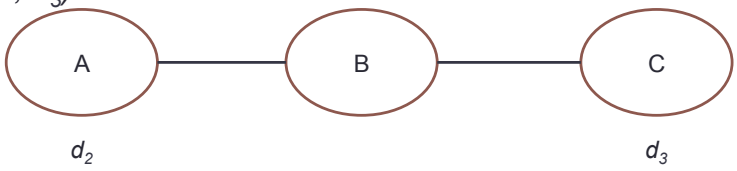
- *put(k, d_0)*



76

Data Currency em DHTs - Problema 2

- Atualizações concorrentes
- *put(k, d_2)*
- *put(k, d_3)*



77

Data Currency em DHTs

- Solução parcial através de versionamento de dados
- Cada réplica possui um número de versão que é aumentado após cada atualização
- Todas as réplicas são recuperadas e ordenadas para selecionar a versão mais recente
- Ainda não resolve o problema de atualizações concorrentes

78

Data Currency em DHTs

- Solução mais completa que considera disponibilidade e concorrência
- Usa um conjunto independente de funções hash que armazena um *timestamp* lógico
- Cada par *(k, data)* é identificado por um *timestamp*
  - Update Management Service (UMS)
- Não é necessário recuperar todas as réplicas para encontrar a mais atual

Referências

- **Principles of Distributed Database Systems.** M. Tamer Özsu, Patrick Valduriez. 3ª edição

GERENCIAMENTO DE DADOS EM REDES P2P

Josiane Bezerra (jbf2)
Samuel Arcoverde (sfa)