Large-scale Pattern Matching

Presented by ZhenlongYuan

Content

- Background
- Algorithm Design
- Evaluation
- Conclusion
- Future Work

Background

- What is Large-scale Pattern Matching ?

C	P	M	_	a	n	n	u	a	l	_	c	o	n	f	e	r	e	n	c	e	_	a	n	n	o	u	n	c	e

window

A large amount of patterns

a	n	n	o	u	n	c	e

a	n	n	u	a	l

a	n	n	u	a	l	l	y

……

Background

- Why do we need Large-scale Pattern Matching ?
  - ✓ IDS/IPS
  - ✓ UTM
  - ✓ Spam filtering system
  - ✓ Virus Scanning System

Background

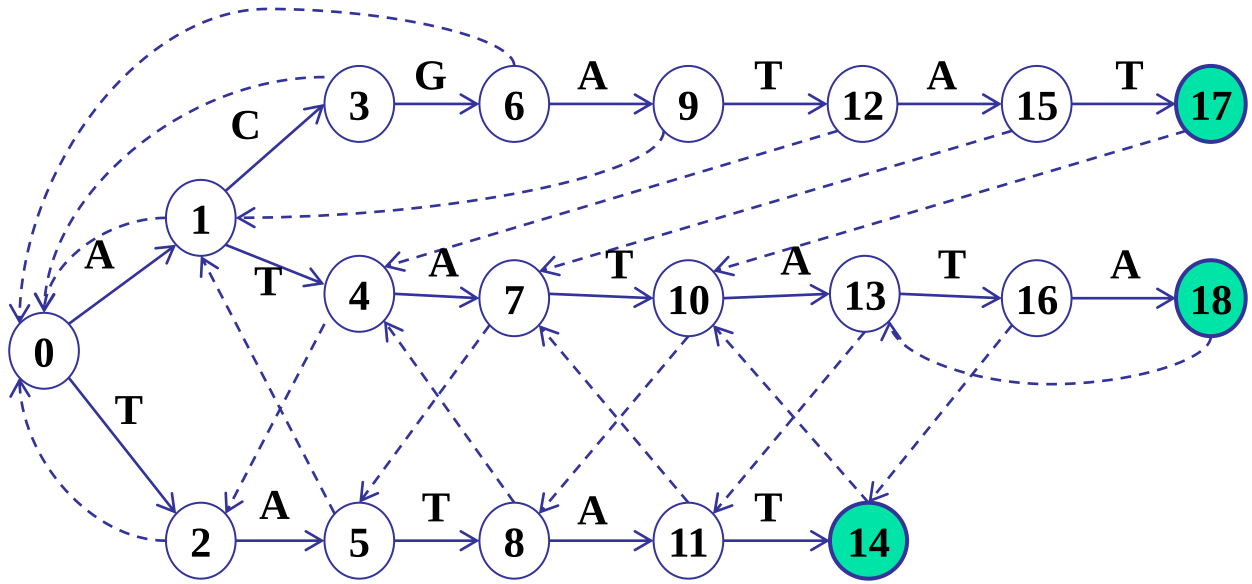
Multi-pattern Matching Algorithms

- Prefix Based
  - ✓ Multiple Shift-And
  - ✓ Aho-Corasick
  - ✓ Based on Aho-Corasick
- Suffix Based
  - ✓ Commentz-Walter
  - ✓ Set Horspool
  - ✓ Wu-Manber
- Factor Based
  - ✓ Multiple BNDM
  - ✓ Set Backward Dawg Matching
  - ✓ Set Backward Oracle Matching(SBOM)

AC Algorithm

P={ATATATA,TATAT,ACGATAT}

- Time complexity is O(n)
  Space complexity is O(|P|x |Σ|)

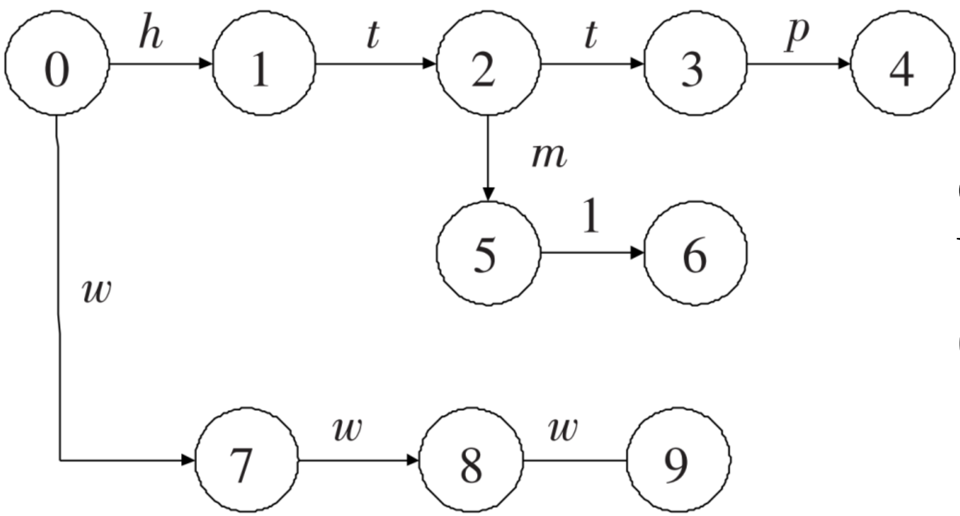


Double-Array Algorithm

T=hthttp

P={http,html,www}

h-1,w-2,t-3,m-4,p-5,l-6



check[base[s]+c]=s

base[s]+c=t

(c is the input character)

int base[];

int check[];

下标	1	2	3	4	5	6	7	8	9	10	11
Base	3	3	0	0	7	4	5	5	–9	–10	–11
Check	0	0	0	0	2	1	6	6	5	7	8
前缀	*h*	*w*			*ww*	*ht*	*htt*	*htm*	*www*	*http*	*html*

WM Algorithm

P={she,hers,his,kiss}

```
static unsigned HASH16(unsigned char *T)
{return (unsigned short) (((*T)<<8) | *(T+1));}
```

B=log|Σ|（2 x lmin x r）

SHIFT

index	shift
...	2
H(sh)	*1*
...	2
H(he)	*0*
H(hi)	*1*
H(er)	*0*
...	2
H(ki)	*1*
H(is)	*0*
...	2

HASH

index	
...	
H(sh)	
...	
H(he)	→ she(1)
H(hi)	
H(er)	→ hers(2)
...	
H(ki)	
H(is)	→ his(3) → kiss(4)
...	

PREFIX

index	id
s	1
h	2,3
k	4

WM Algorithm

T=CPM_annual_conference_announce

P={announce,annual,annually}

SHIFT[Bl]=

字符串	ll	no ou	an	un nc	ua al	ly	nn nu	ce	*
移动距离	1	3	4	1	0	0	2	0	5

HASH[Bl]=

字符串	ce	ly	ua al	*
P中字符串的序号	3	1	2	*
	announce	annually	annual	

Default=M-B+1

CPM_annual_conference_announce

window

CPM_annual_conference_announce

匹配成功1次

CPM_annual_conference_announce

CPM_annual_conference_announce

CPM_annual_conference_announce

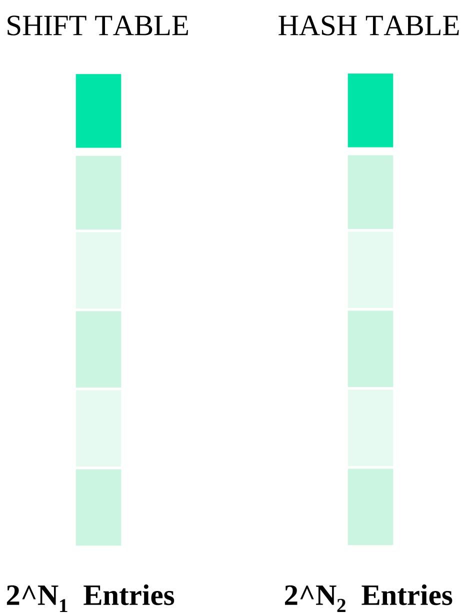
CPM_annual_conference_announce

CPM_annual_conference_announce

CPM_annual_conference_announce

MDH Algorithms (Multi-phase Dynamic Hash)

1.Multi-phase Hash



At its ideal circumstances, if $k^2/2^{(N1+N2)} < k/2^N$, then $R_{mdh} < R_{wm}$

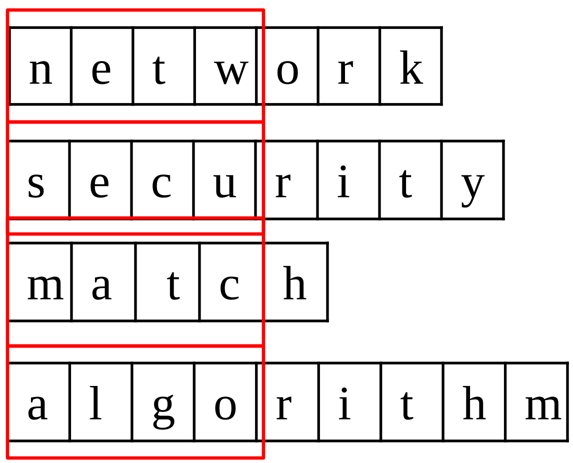
So assume k=100000 and N=20, then N1+N2>37 is OK

h (block) (*(block)) & 0x000FFFFF

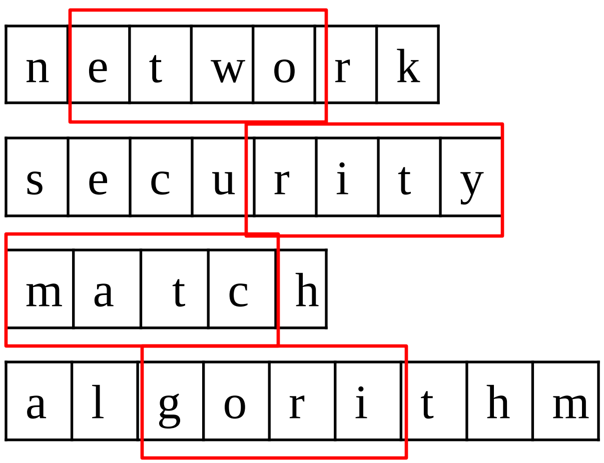
h (block) (((*(block)) <<12) +((*(block +1)) <<8) + ((*(block +2)) <<4) +((*(block +3)) <<12)) & 0x0001FFFF

MDH Algorithm

2.Dynamic-cut Heuristics



Normal WM Algorithm



MDH Algorithm

SRS Algorithm

1. Same_pos:

   Record the relation between adjacent patterns.

2. Same_shift:

   Record the value of skip.

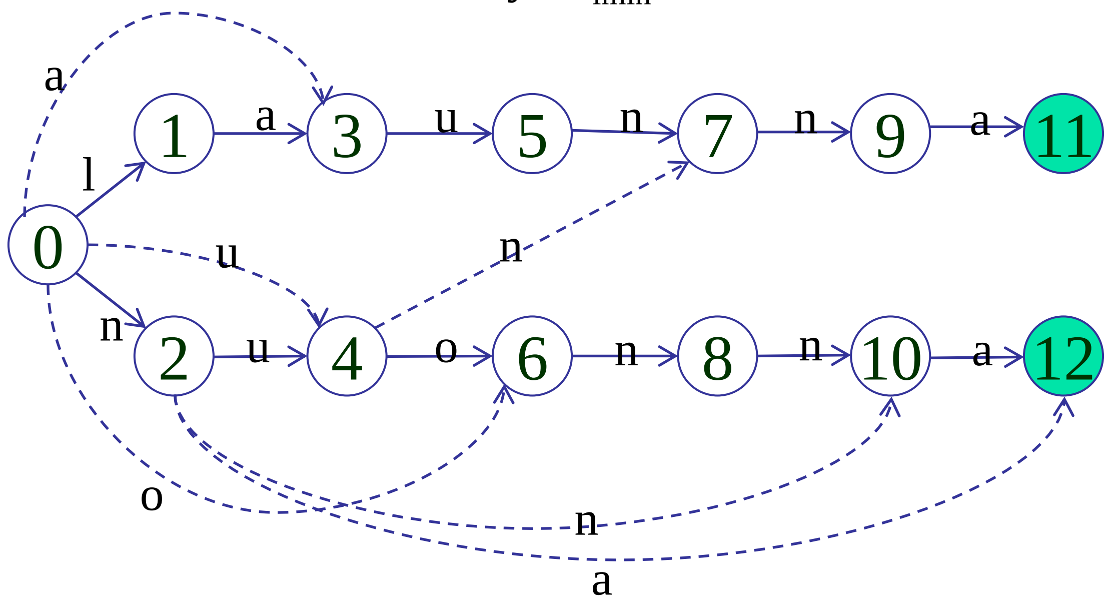
SBOM Algorithm

- Based on Factor Oracle
- Similar with AC Algorithm
- Time complexity is $O(n \times |P|)$
- Space complexity is $O(|P| \times |\Sigma|)$

SBOM Algorithm

T=CPM_annual_conference_announce

P={announce,annual,annually} P_{lmin}={announ,annual}



CPM_annual_conference_announce

CPM_annual_conference_announce

CPM_annual_conference_announce

CPM_annual_conference_announce

CPM_annual_conference_announce

CPM_annual_conference_announce

CPM_annual_conference_announce

Contrast Algorithms

	Advantages	Disadvantages
AC	linear complexity	space consumption
Double-Array	compress the space of FSM	non-skip
WM, MDH and SRS	skip by text window and improve the space utilization	hash collision problem in hash tables
SBOM	skip frequently	space consumption

Algorithm Design

- Based on above algorithms, how to integrate the advantages of them and get rid of their disadvantages?

1. Take advantage of WM and MDH to skip text and solve space consumption

2. Take advantage of AC and Double-Array to solve hash collision problem

→ TMUPM Algorithm

Algorithm Design

1.Multi-phase Hash($N_1 = N_2 + 3$)

Hash Function 1(N_1=26)

$$h_1(block) = \left(*\,(block)\right)\&0x03FFFFFF$$

uchar *shift;

Hash Function 1(N_2 =23)

$h_2(block) =$

$$\left(\left(\left(*\,(block)\right) \ll 15\right) + \left(\left(*\,(block+1)\right) \ll 10\right) + \left(\left(*\,(block+2)\right) \ll 5\right) + \left(\left(*\,(block+3)\right)\right)\right)\&0x007FFFFF$$

typedef struct hashnode {

PAT **next;

uint nodenum;

uchar skip; → TMUPM can still skip after exact match

DOUBLE_ARRAY *arr;

uchar *save;

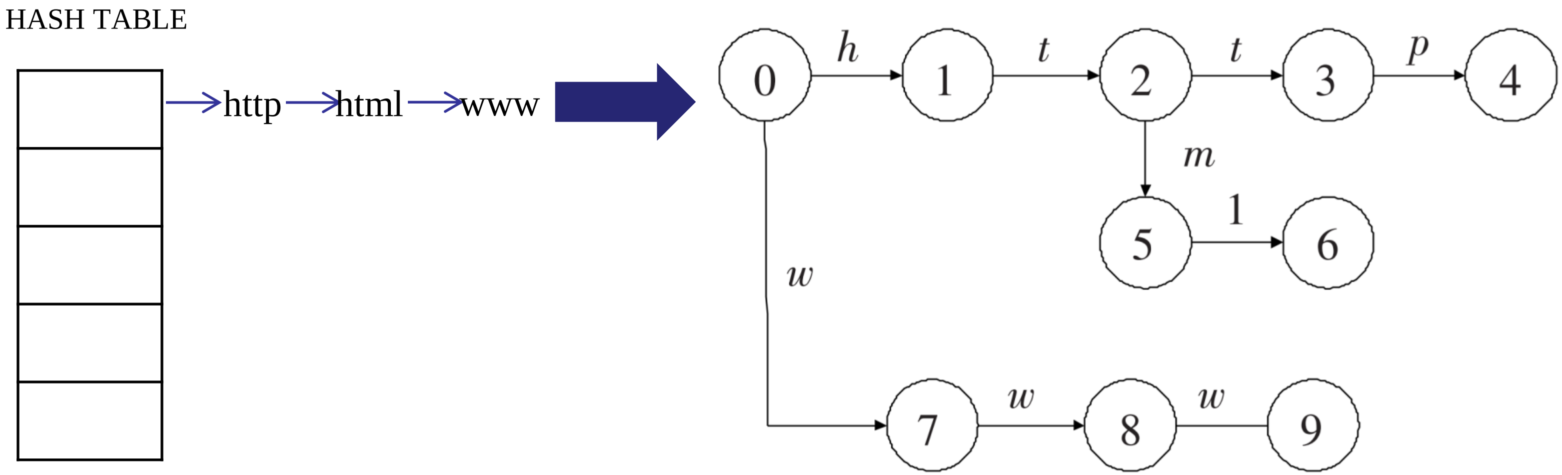
}HASHNODE;

HASHNODE *hashtable;

Algorithm Design

In order to solve the hash collision that happens in hash table.

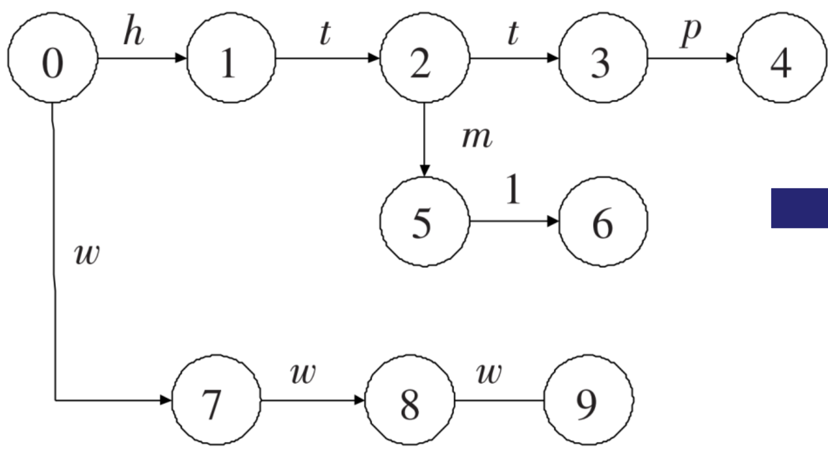
For each entry in hash table, constructing a FSM by using the patterns linked to it.



Algorithm Design

FSM requires large memory, then how to solve it with guaranteeing the algorithm complexity?

FSM

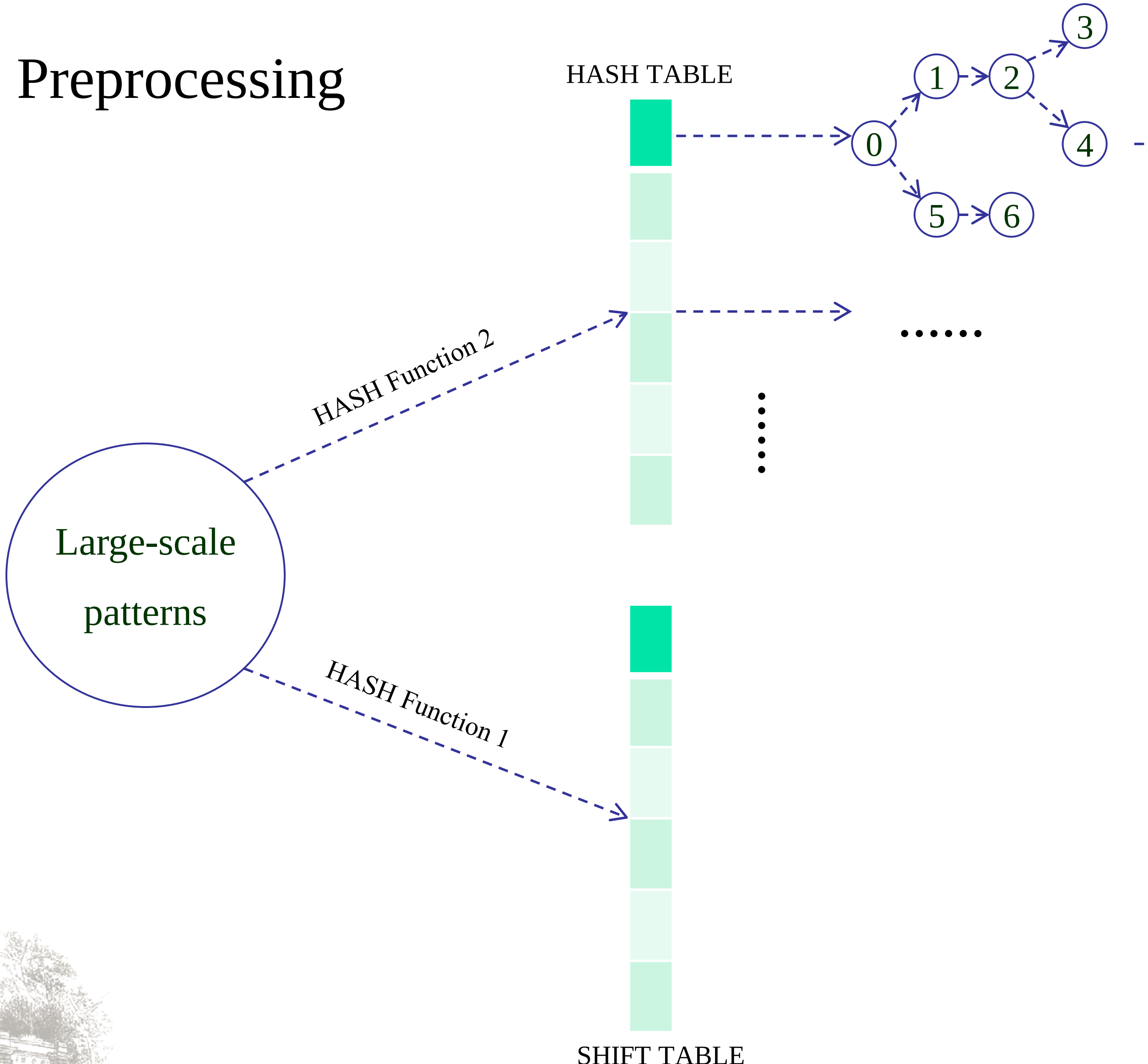


Double-Array

下标	1	2	3	4	5	6	7	8	9	10	11
Base	3	3	0	0	7	4	5	5	−9	−10	−11
Check	0	0	0	0	2	1	6	6	5	7	8
前缀	*h*	*w*			*ww*	*ht*	*htt*	*htm*	*www*	*http*	*html*

Algorithm Design

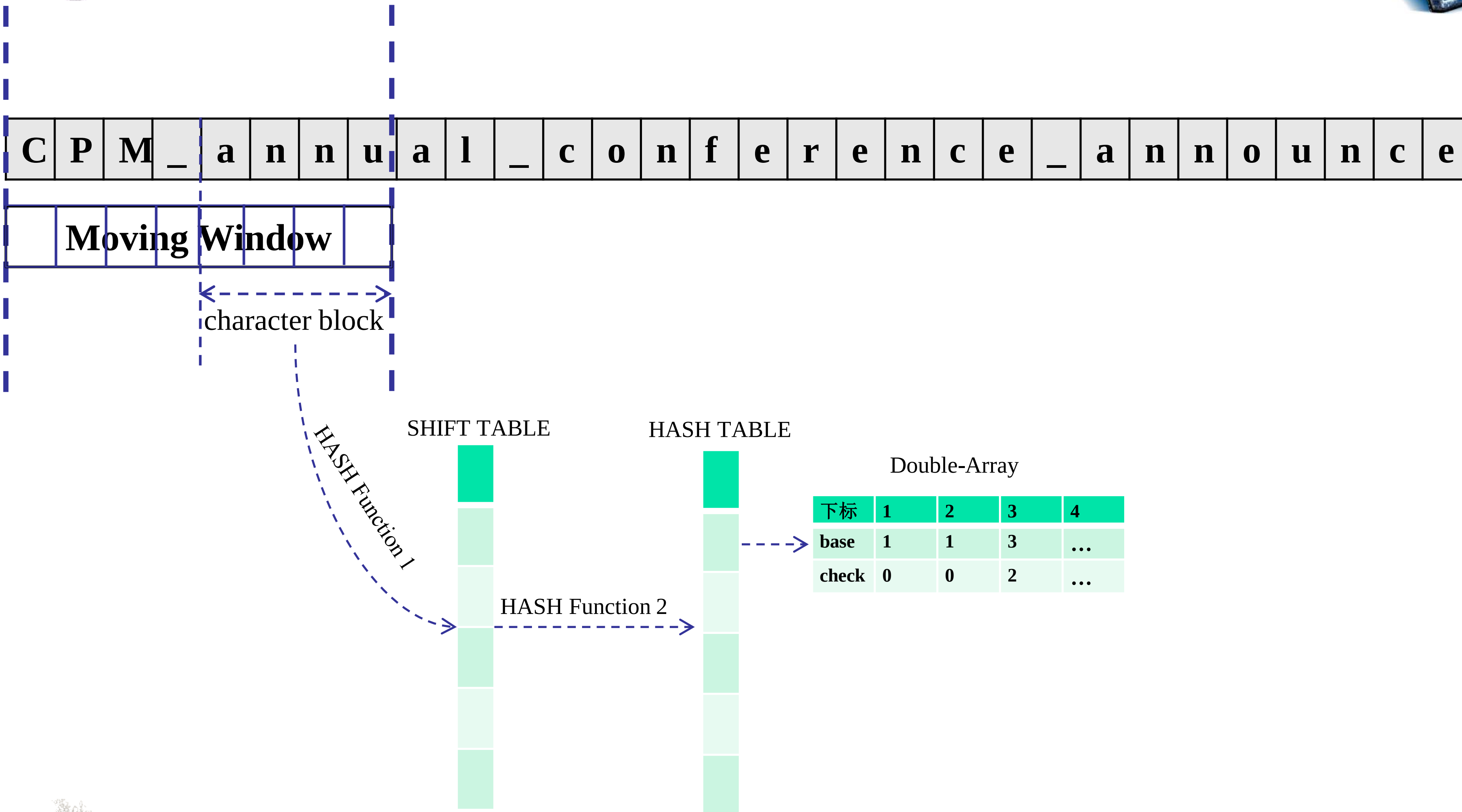
- Preprocessing



Double-Array

下标	1	2	3	4
base	1	1	3	…
check	0	0	2	…

Algorithm Design

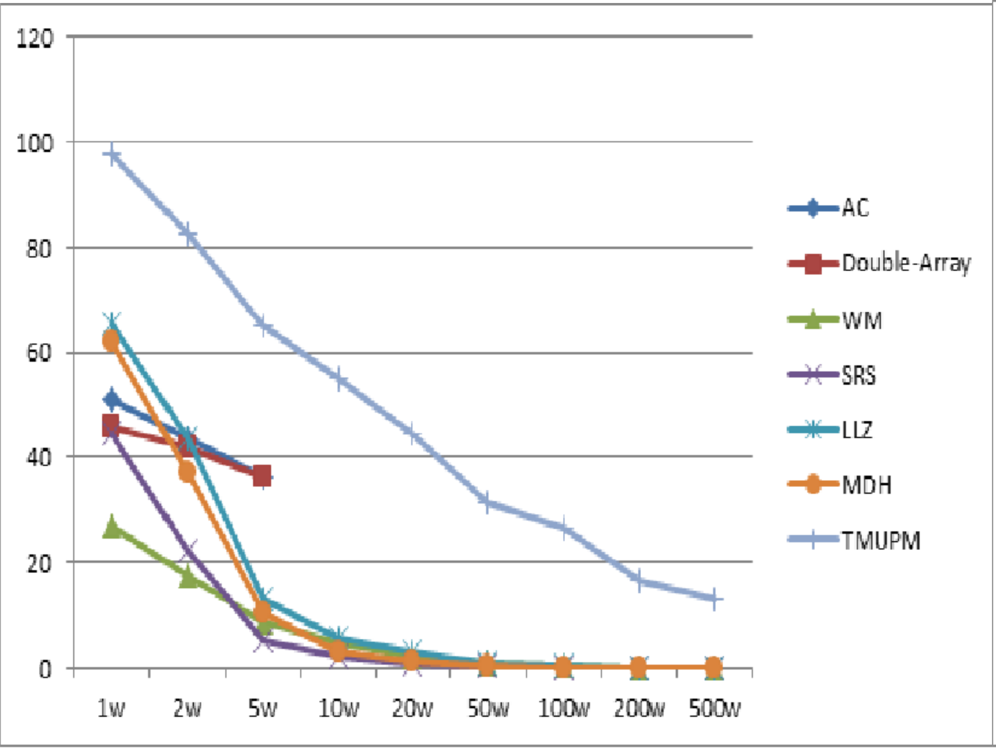


下标	1	2	3	4
base	1	1	3	…
check	0	0	2	…

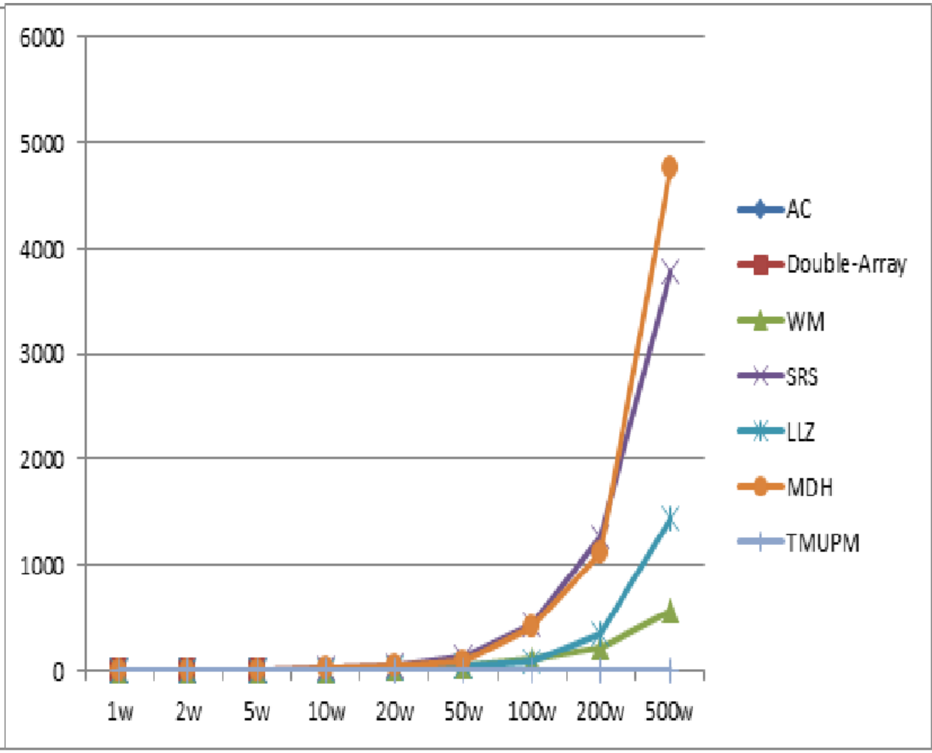
- Matching Process

Evaluation

Speed(MB/s)



Time(s)



Conclusion

- Required space:
  - ✓ The hash functions can control the space consumption by adjusting N_1 and N_2 .
  - ✓ The large-scale pattern set is divided into many small Double-Array structures.
- Matching speed:
  - ✓ TMUPM takes advantage of MDH and Double-Array to skip text and keep linear complexity.
  - ✓ TMUPM can still skip after exact matching in Double-Array.

Conclusion

- Pattern update
  - ✓ Though the preprocessing time of TMUPM may be a bit long, the pattern updating will be easy.
- Perfect for
  - ✓ Whatever the scale, TMUPM is faster than most algorithms.
  - ✓ However the hash collision, TMUPM is faster than most algorithms.

Future work

- 1. Compressing the space of Double-Array
- 2. Relation of N1， N2 and hash functions.

Thank you!

Any Question?