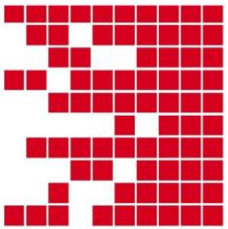


Make

The Global Utilization Cycle of Big Data

Shinji FURUSHO

Turbo Data Laboratories, Inc.

http://turbo-data.co.jp/en/

rev. 3.1

March. 11th, 2019

Contents -1

Contents -2

Contents -3

- Visit our home pages to see more

Preface

To use Big Data is not easy.
To use free combinations of Big Data is impossible at all.
When they become easy,

1. Everyone enjoys Big Data as contents located all over the world.
2. Big Data Markets appear.
3. Services which mash up Big Data like advanced analytics*1, forecasts *2, simulations*2 appear.
4. Technologies depending on Big Data like AI and optimizations improve to the next level.

What prevents to use of Big Data is

1. No mobility.
2. The absence of acceleration methods which can cover every field, subset, and operation.
3. The absence of preemptive multi-tasking technology of the massive parallel system.
4. The absence of methodology to make the global utilization cycle of Big Data.

I solved those problems 1 to 3 above by three technologies named Zap-Over, Zap-In, and Zap-Mass. Moreover, I suggest solving the 4th problem through the combination of these three technologies. These challenges started in 1995. I'm glad to introduce them here.

*1. Current analytics like BI cannot refer or handle Big Data.
*2. Forecasts and simulations need to gather and combine many kinds of Big Data, but that is almost impossible.

Make The Global Utilization Cycle of Big Data;

Abstract (1/4)

What prevents to use Big Data is

1. No mobility.
2. The absence of acceleration methods which can cover every field, subset, and operation.
3. The absence of preemptive multi-tasking technology of the massive parallel system.
4. The absence of methodology to make the global utilization cycle of Big Data.

The 1st problem has solved by the technology named ***Zap-Over***.

By Zap-Over, we can create a ***Big Data's Browser can handle trillions of records across clouds***.

1. It can make a virtual union table immediately from arbitrary picked up tables
   having trillions of records, having different schemas, and located in many clouds.
2. It can browse that virtual union table both in record order view and sorted order view of any field.
3. It can make a search to the virtual union table by any field's value quickly.

This Zap-Over comes possible by a new sorting technology named "***Target Sort***".
It can make sort with about O(log(n)) steps, because it sorts only the required point.
Because of that, Zap-Over can handle trillions of records easily.

Because Zap-Over uses a sort, it works well to any combination of data unlike legacy technologies like B-Tree.
So, Zap-Over can handle arbitrary combination of tables.

Zap-Over's first trial version is running since 2013 at Tokyo Regional Taxation Bureau for tracing worldwide money transactions.

Abstract (2/4)

What prevents to use Big Data is

1. No mobility.
2. The absence of acceleration methods which can cover every field, subset, and operation.
3. The absence of preemptive multi-tasking technology of the massive parallel system.
4. The absence of methodology to make the global utilization cycle of Big Data.

The 2nd problem has solved by the technology named ***Zap-In***.

By Zap-In, we can create a ***Big Data's Spreadsheet with Relational Algebra support***.
It can handle usual tables and virtual join tables having up to 2 billion records.
With it, we can do every operation expected to Big Data's Spreadsheet with Relational Algebra support,
edit/calculate/search/sort/tabulation/type conversion/set operations/categorization/matching/union/join/extract …

That becomes possible by the data structure called Zap-In TR and many algorithms whose inputs/outputs are Zap-In TR.
Because Zap-In TR has dual functions:

1. It is data from one side,
2. It is a base of efficient algorithms from another side.

Then we can use Zap-In TR as data, and can use it as a base of so many algorithms which can every field, subset, and operation.

Zap-In has been used from 2002 in many sites aerospace, nuclear plant, credit card company, huge chemical industry, and others.

Abstract (3/4)

What prevents to use Big Data is

1. No mobility.
2. The absence of acceleration methods which can cover every field, subset, and operation.
3. The absence of preemptive multi-tasking technology of the massive parallel system.
4. The absence of methodology to make the global utilization cycle of Big Data.

The 3rd problem has solved by the technology named ***Zap-Mass***.

By Zap-Mass, we can create a ***Big Data's Massive Parallel RDB with preemptive multi-tasking***.
To handle true Big Data, we have to use massive parallel system.
However, it was impossible to run jobs in massive parallel system over preemptive multi-tasking environment until now.
Then the system is unreliable, inefficient, and not easy to manage/use.

To enable preemptive multi-tasking over massive parallel system, we hope algorithms between nodes to have following features.

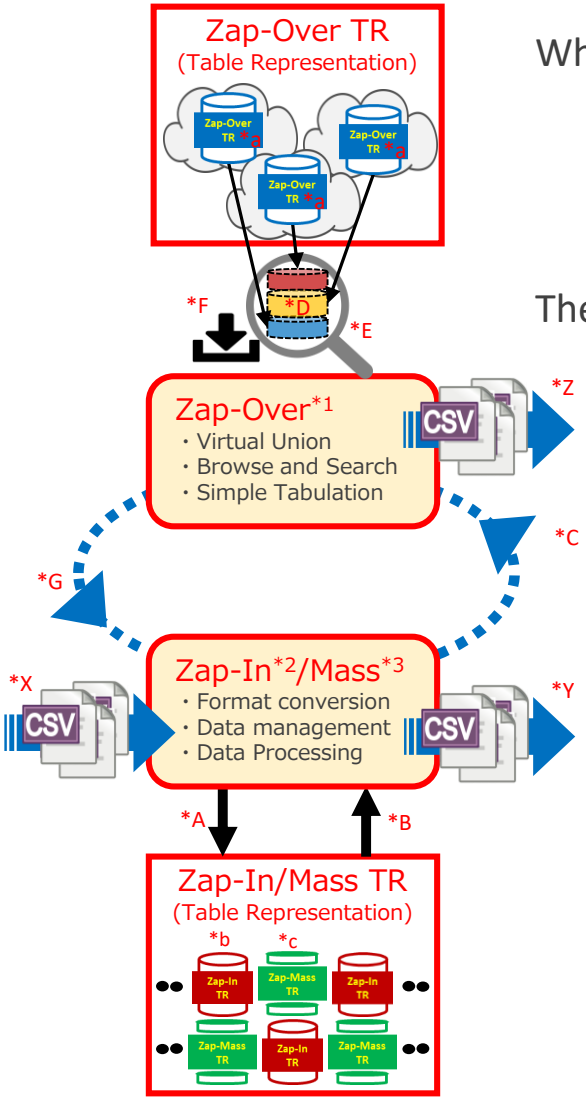
1. *Universality*:
2. *Symmetry*:
3. *Independence from processing order*:
4. *Divisibility of communication packets*:

When Zap-In runs inside each node, then inter-node algorithms have said four features.

Moreover, good inter-node architecture is necessary. Here I propose multi-ring architecture having following features.

1. *Symmetry*:
2. *Two directions of communication paths*:
3. *Divisibility of communication paths*:

Abstract (4/4)



What prevents to use Big Data is

1. No mobility.
2. The absence of acceleration methods which can cover every field, subset, and operation.
3. The absence of preemptive multi-tasking technology of the massive parallel system.
4. The absence of methodology to make the global utilization cycle of Big Data.

The 4rd problem has solved by the combination of ***Zap-Over/In/Mass***.

The contrast between these 3 systems

Zap-Over is a technology for Big Data's ***Browser*** can handle trillions of records across clouds.
- It is enabled by the theme 1: reducing sorting steps down to about O(log(n)).
- It can make a virtual union table from tables having trillions of records distributed across clouds.
- It can browse and search the virtual union table smoothly and quickly.
- It bases upon on-disk technology.

Zap-In is a technology for Big Data's ***Spreadsheet*** with relational algebra support.
- It is enabled by the theme 2: free cascading of arbitrary algorithms/subsets.
- It can handle up to 2 billion records.
- It bases upon in-memory technology.

Zap-Mass is a technology for Big Data's ***RDB*** every one can use.
- It is enabled by the theme 3: preemptive multitasking of massive parallel system.
- It can handle tables having trillions of record.
- It runs stably, reliably, efficiently, quickly, flexibly and it is easy to use.
- It bases upon massive parallel technology.

Technology name (Data Structure Name)	Architecture	Enabled Application	Theme	Products/Achievements
Zap-Over (Zap-Over TR) 	On Disk	Big Data's *Browser* can make union tables having trillions of records distributed across clouds	To *reduce sorting steps down to about O(log(n))*	Tokyo Regional Taxation Bureau
Zap-In (Zap-In TR) 	In Memory	Big Data's *Spreadsheet* with Relational Algebra support	To enable *free cascading of any algorithm of any subset*	• Sakura's Analytics • I licensed patents to SAP/NEC/Fujitsu/…
Zap-Mass (Zap-Mass TR) 	Massive Parallel	Big Data's *RDB* everyone can use	To enable *preemptive multi-tasking of massive parallel system*	-

How the Big Data's Globally Connected Cycle Works

1. Zap-Over
(Zap Over Network)

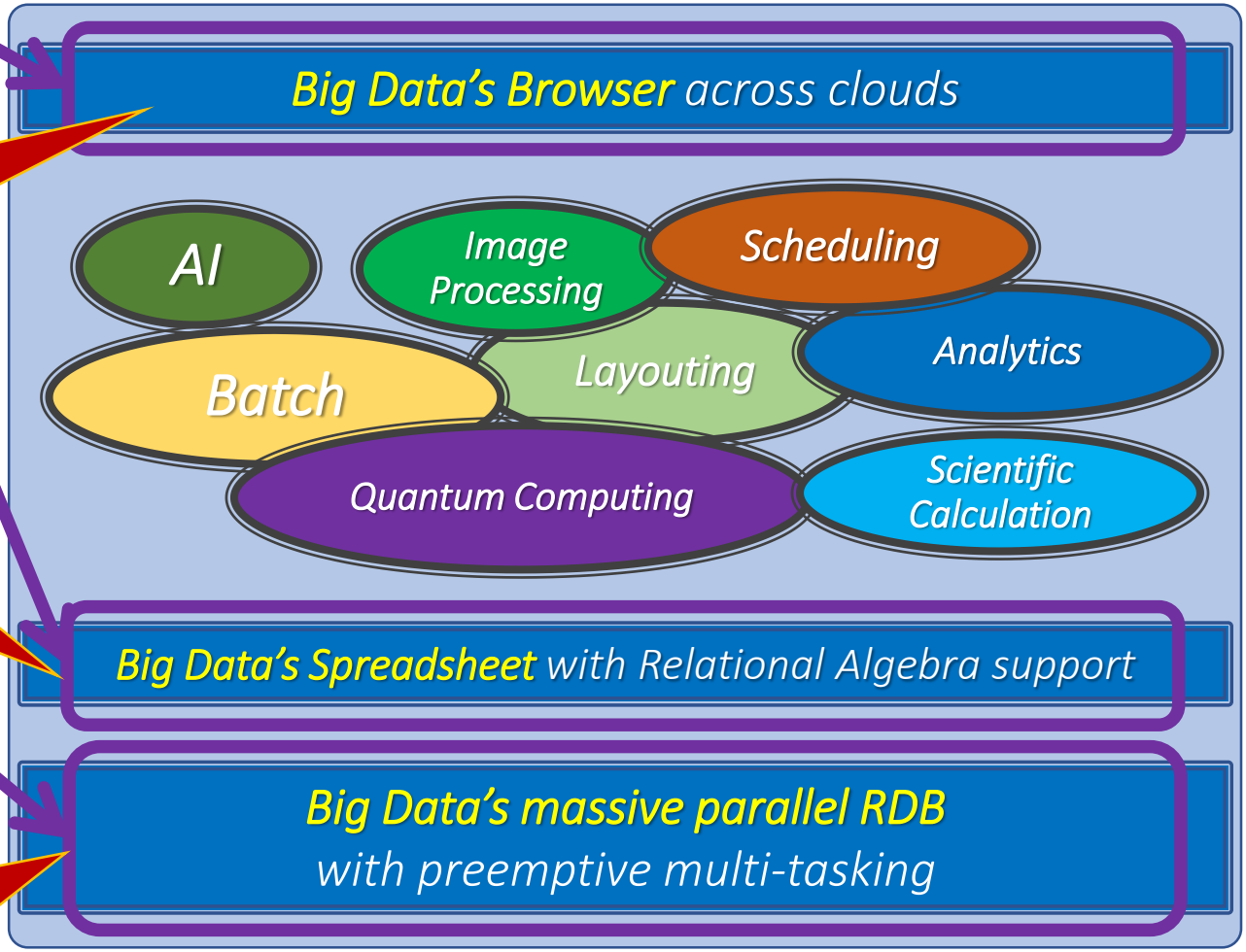
The Big Data's **Browser**, allows non-professional users to fetch Big Data across clouds.

2. Zap-In
(Zap In Memory)

The Big Data's **Spreadsheet** allows non-professional users to process Big Data.

3. Zap-Mass
(Massive Parallel)

The Big Data's massive-parallel **RDB** with preemptive multi-task, allows non-professional users to process Big Data.



Make The Global Utilization Cycle of Big Data;

Zap-Over: The Big Data's Browser across Clouds

Zap-Over: Zap Over Network

Come to our home page to see
Zap-Over demo.

http://turbo-data.co.jp/en/

Zap-Over: The Big Data's Browser across Clouds

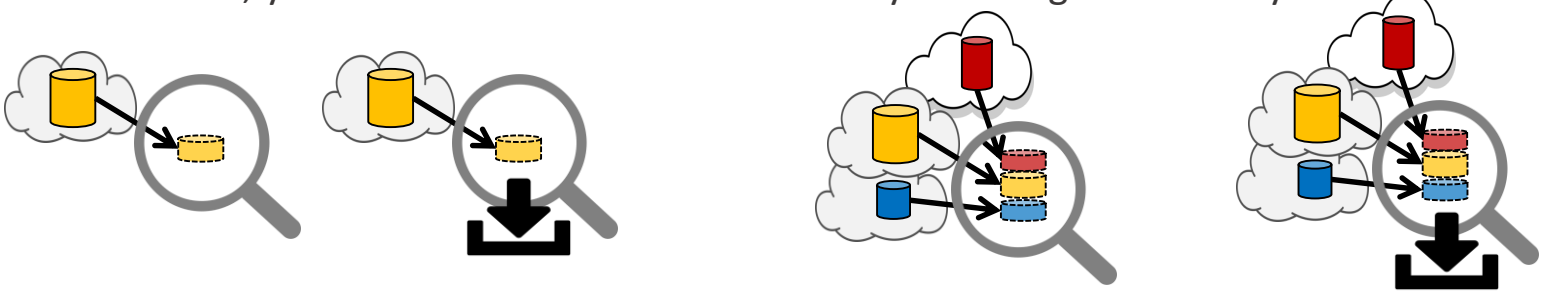
"No Mobility" problem arises because

1. You cannot access it without a system dedicated for it.
2. You cannot download it.
3. You cannot combine arbitrary Big Data.

Zap-Over gives "mobility" to Big Data, it enables

The Big Data's Browser can handle trillions of records across clouds

1. You can browse/search Big Data Tables only with a software: "The Big Data's Browser."
2. You can download Big Data Tables by selecting out where you need by the browser.
3. You can make a Big Data's virtual union table from arbitrary Big Data tables across clouds by the browser.
   Moreover, you can browse/search the virtual union table with it.
   Moreover, you can download the union table by selecting out where you need with it.



Zap-Over: The Big Data's Browser across Clouds

fig. 1. Hopping combined Big Data Table

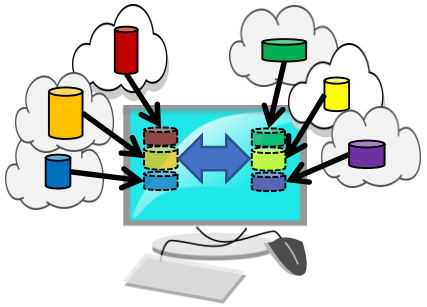
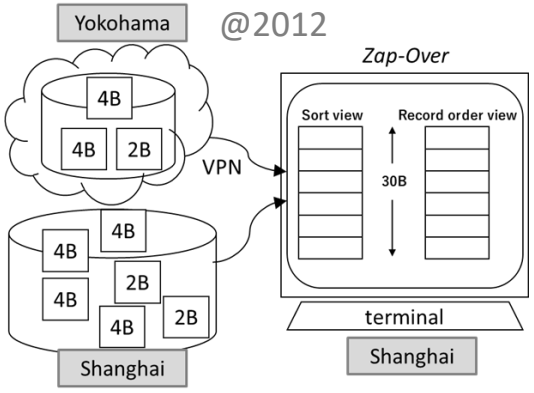


fig. 2. Two way browse:
Sort View/Record Order View



For what the Big Data's Browser is?

For User Side:

1. Hopping to another Big Data immediately → fig 1
2. Browsing the Big Data smoothly → fig 2
   Sort Views of each field / Record Order View
3. Searching the Big Data quickly → fig 3
4. Downloading a part of Big Data → fig 4

For Cloud Side:

1. You can provide Big Data service by <u>just putting a file</u>

fig. 3. Searching of Big Data

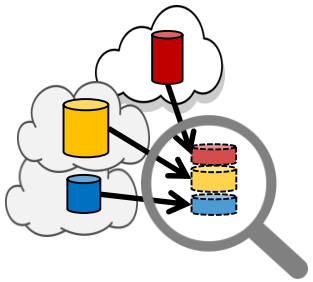
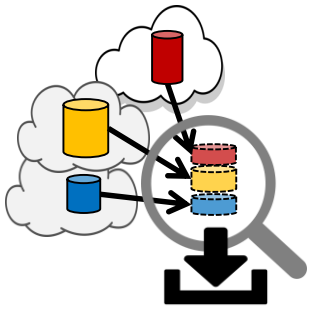
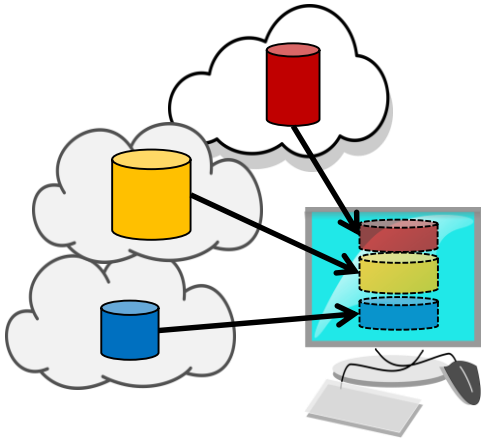


fig. 4. Downloading of Big Data



Zap-Over: The Big Data's Browser across Clouds

What is the Big Data's Browser?



1. It can create a virtual union table from any Big Data table
   - Having Trillions of records,
   - Having different Schemas and
   - Distributed across Clouds.

2. It gives a two-way view of that virtual union table
   - Record order view,
   - Sort view by any field
     → Sort view ***enables search***

Original Table 0

0	F	8
1	M	7
2	M	6
3	F	7

Original Table 1

0	M	6
1	F	8
2	M	8
3	F	7

Record Order View

0	F	8
1	M	7
2	M	6
3	F	7
4	M	6
5	F	8
6	M	8
7	F	7

Sort View (by Gender)

			RecNo
0	F	8	0
1	F	7	3
2	F	8	5
3	F	7	7
4	M	7	1
5	M	6	2
6	M	6	4
7	M	8	6

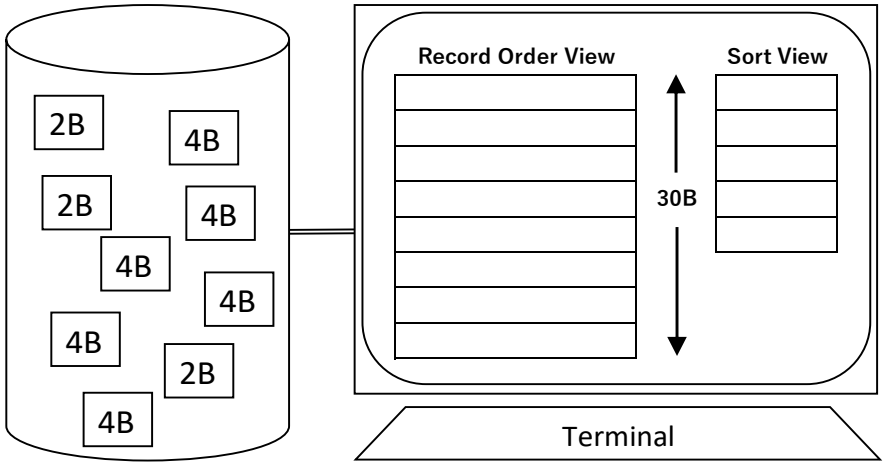
Sort View (by Age)

			RecNo
0	M	6	2
1	M	6	4
2	M	7	1
3	F	7	3
4	F	7	7
5	F	8	0
6	F	8	5
7	M	8	6

Zap-Over: Big Data's Browser Demonstration (1/3)

Sample Data @2012

File number: 1	File number: 2
File name: Sc4billionx6.D5A	File name: Sc2billionx6.D5A
File size: 842,000,113,664 bytes	File size: 420,735,533,536 bytes
Number of rows: 4,000,000,000	Number of rows: 2,000,000,000
Number of fields: 6	Number of fields: 6
Field 1. Integer 64 bit	Field 1. Integer 64 bit
Name: int1K	Name: int1K
Cardinality: 1,000	Cardinality: 1,000
Field 2. Integer 64 bit	Field 2. Integer 64 bit
Name: int2G	Name: int2G
Cardinality: 2,000,000,000	Cardinality: 1,000,000,000
Field 3. Double	Field 3. Double
Name: Dbl1K	Name: Dbl1K
Cardinality: 1,000	Cardinality: 1,000
Field 4. Double	Field 4. Double
Name: Dbl2G	Name: Dbl2G
Cardinality: 2,000,000,000	Cardinality: 1,000,000,000
Field 5. String	Field 5. String
Name: Str1K	Name: Str1K
Cardinality: 1,000	Cardinality: 1,000
Field 6. String	Field 6. String
Name: Str2G	Name: Str2G
Cardinality: 2,000,000,000	Cardinality: 1,000,000,000



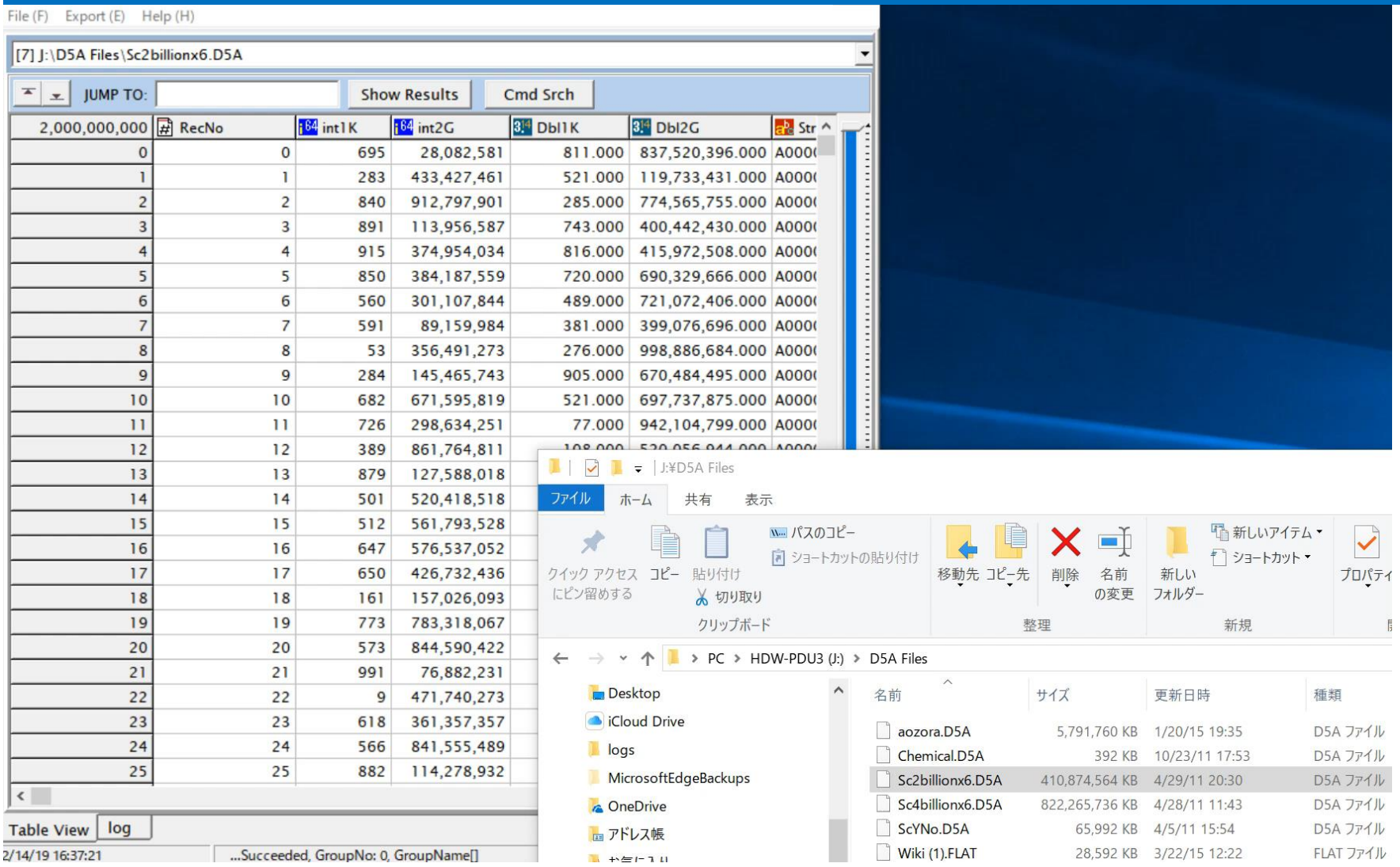
Zap-Over: Big Data's Browser
Demonstration (2/3)

1. It makes a virtual union table having 30B records.
2. Next, it shows scroll of that 30B in the record order view.
3. Next, it shows a sort view and operations (search, etc.) in it.
   See the hit result having 9B is sorted and shown immediately.
   See the sort view is linked to the record order view.

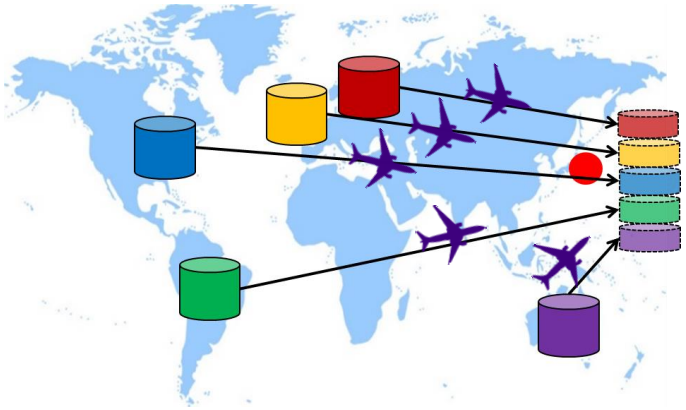
Come to our home page to see Zap-Over demo.

http://turbo-data.co.jp/en/

Zap-Over: Big Data's Browser Demonstration (3/3)



Zap-Over: The power of the algorithm



The power of the algorithm

As a performance example, Zap-Over can sort with about O(log(n)) steps where n is the count of records.

> *With it, we can browse and search any virtual union of Big Data, which have trillions of records and distributed across clouds.*

It has been used in the real system at Tokyo Regional Taxation Bureau since 2013 to seek global money laundering: '4S System'.

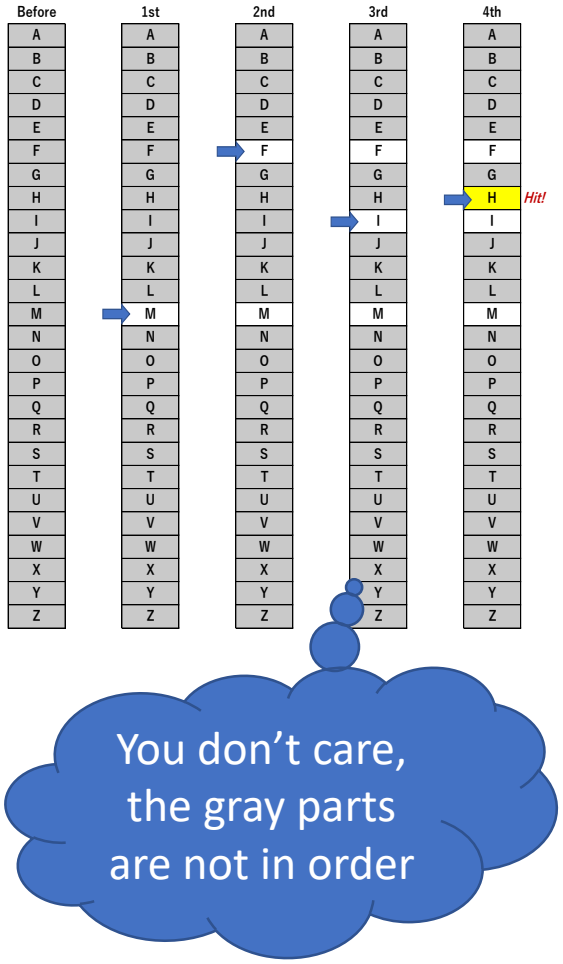
It could show only x1,000 total performance than a former system made of SQL Server that took 15~20 minutes each search; allowed 1~2 persons to use simultaneously.

However, it could have shown much more performance if it didn't need to compress HDD and it could use SSD.

Zap-Over: "Target Sort" Can Sort Trillions.

Why is Big Data's Browser possible?

I have developed "***Target Sort***," that can sort trillions immediately.
Because of it, I can sort any union of one-dimensional arrays instantly.
When I can sort, I can browse/search at once.
So, Big Data's Browser comes true.



Why is the sorting of trillions possible?

Think about the case you use a dictionary. Every article in the dictionary is in order.

However, you don't care articles are not in order, which you don't use.

For example, if a dictionary has 1,000,000,000,000 articles inside, we use log2(1,000,000,000,000) = 40 articles for one search. You don't care other articles are not sorted.

For another example, for a browser's sort view, we need only 100 articles at most at one time.

The target sort, unlike other sorts, outputs the pinpoint where you need. So, it can return an answer immediately unlike others.

I show the target sort's algorithm in the following pages.

Zap-Over: The Target Sort's algorithm

An algorithm example: Target Sort – 0 Initial State

Terminology
- SVL: Sorted Value List
- Aggr: Aggregation of occurrences
- INV: Inverted record number array

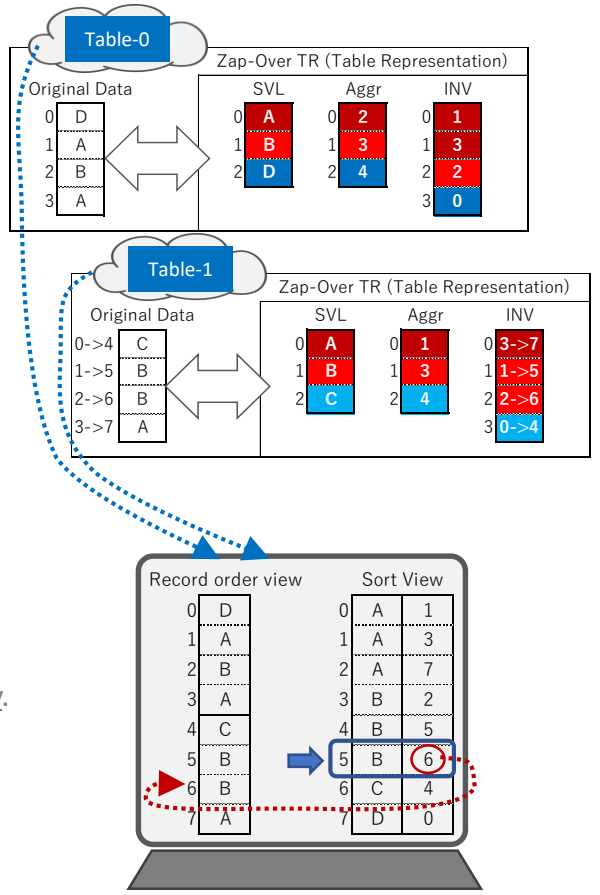


Fig 1. target sort

Cloud side:

In the figure, there is an original data array in each cloud. (See the terminology also)

The cloud side should make SVL/Aggr/INV arrays from the original data before providing
- SVL by extracting values included in each array.
- Aggr by counting up each value's occurrences first and aggregation next.
- INV by sorting the original data array with its subscripts.

Then each element in Aggr means, how many values (records) appears less than or equal to the value in the SVL put at the corresponding position. That is the crucial point enabling the target sort.

At last, the cloud side packs up Original Data/SVL/Aggr/INV into "one Big Data file" usually.

Terminal side:

The terminal in the figure selects arbitrary Big Data files across clouds and open them.

To show the record order view:

Just access the original data arrays adjusting their subscripts at reading out time. (No need to rewrite/overwrite)

To show the sort view:

That is the target sort! See the following pages.

Zap-Over: The Target Sort's algorithm

An algorithm example:
Target Sort – 1
Walk Through

1. Seek **belonging value**
2. Seek **belonging table**
3. Calculate the **offset in the Table**
4. return **INV value**

1. Walking through

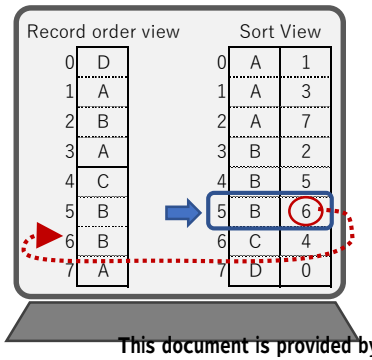
Try to identify sort view's record #5 (Fig. 1)

The goal of target sort is to return the record order view's record number corresponding to the given sort view's record number (this time it is #5).

We can know how many records are there having values less than or equal to any value in SVL, by reading out the element in Aggr at the corresponding position of a value in SVL.

(Following steps is for the case of just after a new record number #5 in the sort view given. Note giving previous/next record number of the sort view, after you got one, is far easier.)

(Sort view's record number = 5)
Step 1. Seek belonging value → belonging value = "B"
Step 2. Seek belonging table → belonging table = Table-1
Step 3. Calculate the offset in the table → Offset = 1
Step 4. return INV value → record order view's record number = 6



Zap-Over: The Target Sort's algorithm

An algorithm example:
Target Sort – 2
Decide belonging value

1. Seek **belonging value**
2. Seek **belonging table**
3. Calculate the **offset in the Table**
4. return **INV value**

2. Each Step

Try to identify sort view's record #5 (Fig. 1)

We can know how many records are there having values less than or equal to any value in SVL, by reading out the element in Aggr at the corresponding position of a value in SVL.

I show the steps to get the sort view's record #5.
These step's goal is to get the record order view's record number corresponding to the sort view's record number #5. (See Fig. 1 at the above page.)
(Following steps is the case of when a new sort view's record number #5 has given.)

Step 1. Seek belonging value → "B"
We can decide which value the sort view's record number #5 belongs, by a similar way of the bisection search, by upping/downing the reference value got from SVLs.
This case the belonging value is "B."
Step 1 needs about log2(m0)+log2(m1) times trials. (m0/m1: size of SVL in Table 0/1)

Step 2. Seek belonging table → Table-1
We can know how many records exist having values less than "B" in all tables.
Moreover, we can know how many records exists having "B" in each table.
Then we can know to which table the sort view's record number #5 belongs.

Step 3. Calculate offset → Offset = 1
Table-1's "B" starts from sort view's record #4, then the offset = #5 - #4 = 1.

Step 4. return INV value → record order view's record number = 6
Look up Table-1, "B" starts at 1 in INV.
Because the offset is 1, the position in INV becomes 1 + 1 = 2.
We get INV[2] = 2; Then adding total record count before Table-1: 4 to it.
The answer is 2 + 4 = 6;

Zap-Over: "Multi-Value" helps many kinds of Search

Multi-Value helps many kinds of Search

For full text search → Table 1

For complex condition search → Table 2

Table 1. "Name" and Multi-Value

Name	Multi Value
Beth	B,E,T,H,BE,ET,TH,BET,ETH,BETH
Ann	A,N,AN,NN,ANN
Tom	T,O,M,TO,OM,TOM

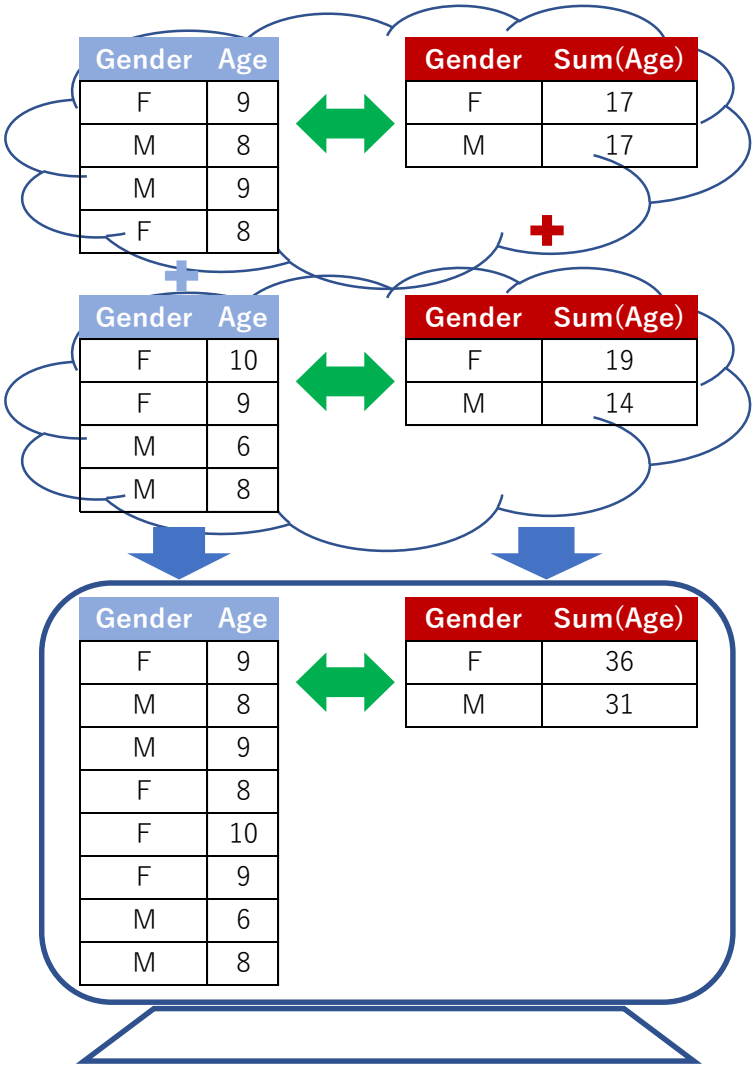
Table 2. "Name" and Multi-Value

Gender	Age	Multi Value
F	9	B,X,Y
M	8	C,Y,Z
M	9	D,Z
F	8	A,X

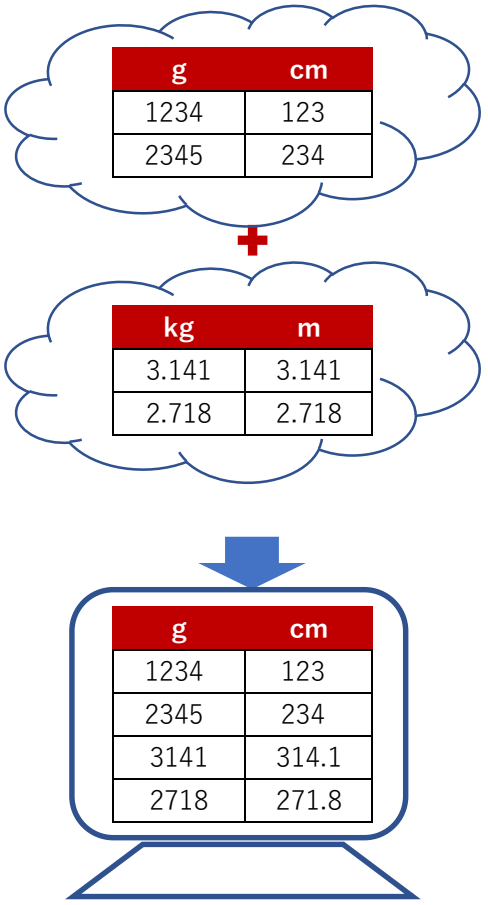
Condition	Encoding
"F" and 8 →	A
"F" and 9 →	B
"M" and 8 →	C
"M" and 9 →	D
A or B →	X
B or C →	Y
C or D →	Z

Zap-Over: Other possible functions

Tabulation



Unit Conversion



Zap-Over works with Zap-In

fig. 1. Zap-Over for Zap-In

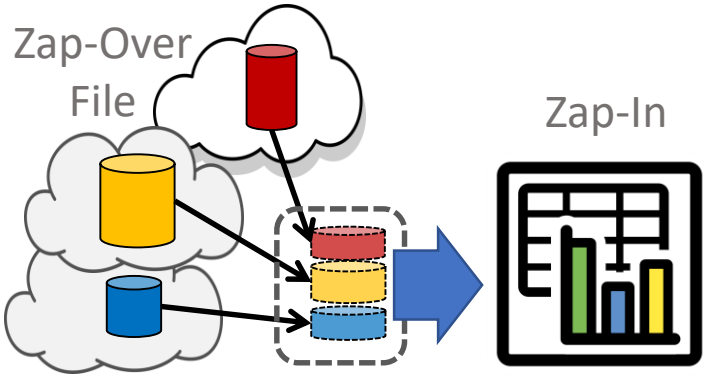
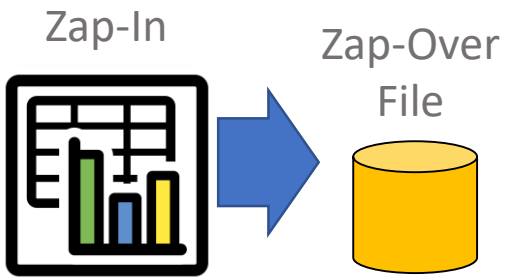


fig. 2. Zap-In for Zap-Over



Zap-Over and Zap-In works together

Zap-Over for Zap-In:

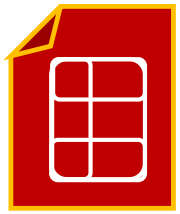
Gathering Big Data across clouds for Zap-In → fig 1

Zap-In for Zap-Over:

Making Zap-Over's Big Data File → fig 2

Zap-Over: About Patents

Several Patents are Pending



Make The Global Utilization Cycle of Big Data;

Zap-In: The Big Data's Spreadsheet with Relational Algebra Support

Zap-In: Zap In Memory

Come to our home page to see Zap-In demo.

http://turbo-data.co.jp/en/

Make The Global Utilization Cycle of Big Data;

Zap-In: What enables Big Data's Spreadsheet with Relational Algebra support

"The absence of acceleration methods can cover every field/subset/operation" is caused by 5 reasons.

1. Indexes such as hash and B-Tree are not attached to every field.
2. Indexes such as hash and B-Tree are not valid to subsets.
3. Indexes such as hash and B-Tree are available only for search.
4. Indexes such as hash and B-Tree use CPU time to create/maintain themselves.
5. Indexes such as hash and B-Tree use memory/storage.

Zap-In solved above 5 problems, and it enables

The Big Data's Spreadsheet with Relational Algebra support.

1. It can handle usual tables; virtual join tables having up to 2 billion records.
2. It can do every operation expected to Big Data's Spreadsheet with Relational Algebra support,
   edit/calculate/search/sort/tabulation/type conversion/
   set operations/categorization/matching/union/join/extract …

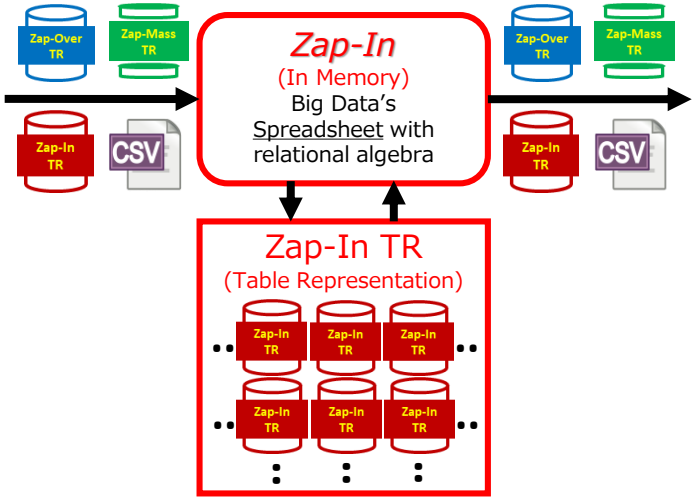
For What "Zap-In: Big Data's Spreadsheet" is?

For Interactive Batch Processing.

1. Interactive Analytics of Big Data.
2. Interactive Data Cleansing / Data Transformation / Data Validation (inspection) …
3. Batch (including created ones as keyboard macro) BOM/MRP/…

For embedding into a system.*A (the right figure)

1. It imports Zap-Over/In/Mass TR or CSV.
2. It does checking/transformation/calculation/matching/so on,
   to the imported data looking up local DW.
   If necessary, it saves this data in its local DW.
3. It exports process results; data extracted from its local DW,
   within the format of Zap-Over/In/Mass TR or CSV.



*A. The fields Zap-In has been used are
aerospace/nuclear/university/medical_research/
chemical_plant/compute_factory/communication_industry/
financial/marketing/traffic/… fields. (See the figure left)

How Zap-In solved said 5 legacy problems

The features of the algorithms of Zap-In are very favorable.

1. *Homogeneity of fields; subsets*: Every algorithm runs on any field with any subset in the same way.
2. *Connectivity of algorithms*: Possible to cascade almost every algorithm freely.
   Because the inputs/outputs of these algorithms are always Zap-In TR.
3. *Minimum CPU Cost*: Almost all algorithms don't use CPU to create and maintain acceleration mechanism.
4. *No memory/storage*: All algorithms don't require memory/storage to keep the acceleration mechanism.
5. *Affinity for parallel processing*: Affinity for parallel processing.

Homogeneity of fields; subsets

Connectivity of algorithms

Minimum CPU Cost; No Memory/Storage

Legacy (Indexes are attached to a few)

Table 1: Salt Concentration and Light Transmittance

Salt Concentration (%)	Transmittance (%T)				
	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5
0	77.23	74.50	64.88	75.27	54.66
3	85.23	92.82	78.91	60.71	57.96
6	88.39	100.05	73.66	66.51	64.54
9	80.71	100.05	68.29	64.91	52.96
12	82.66	117.18	71.01	56.91	46.95
15	72.55	115.40	65.72	66.03	55.38

Without Index

With Index

Algorithm A

Algorithm B

...

Algorithm Z

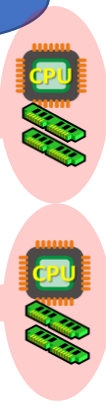
Zap-In TR/
Zap-Mass TR

Zap-In's case
(every field has acceleration mechanism)

Table 1: Salt Concentration and Light Transmittance

Salt Concentration (%)	Transmittance (%T)				
	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5
0	77.23	74.50	64.88	75.27	54.66
3	85.23	92.82	78.91	60.71	57.96
6	88.39	100.05	73.66	66.51	64.54
9	80.71	100.05	68.29	64.91	52.96
12	82.66	117.18	71.01	56.91	46.95
15	72.55	115.40	65.72	66.03	55.38

Homo-geneity



Legacy (Indexes are attached to a few)

Table 1: Salt Concentration and Light Transmittance

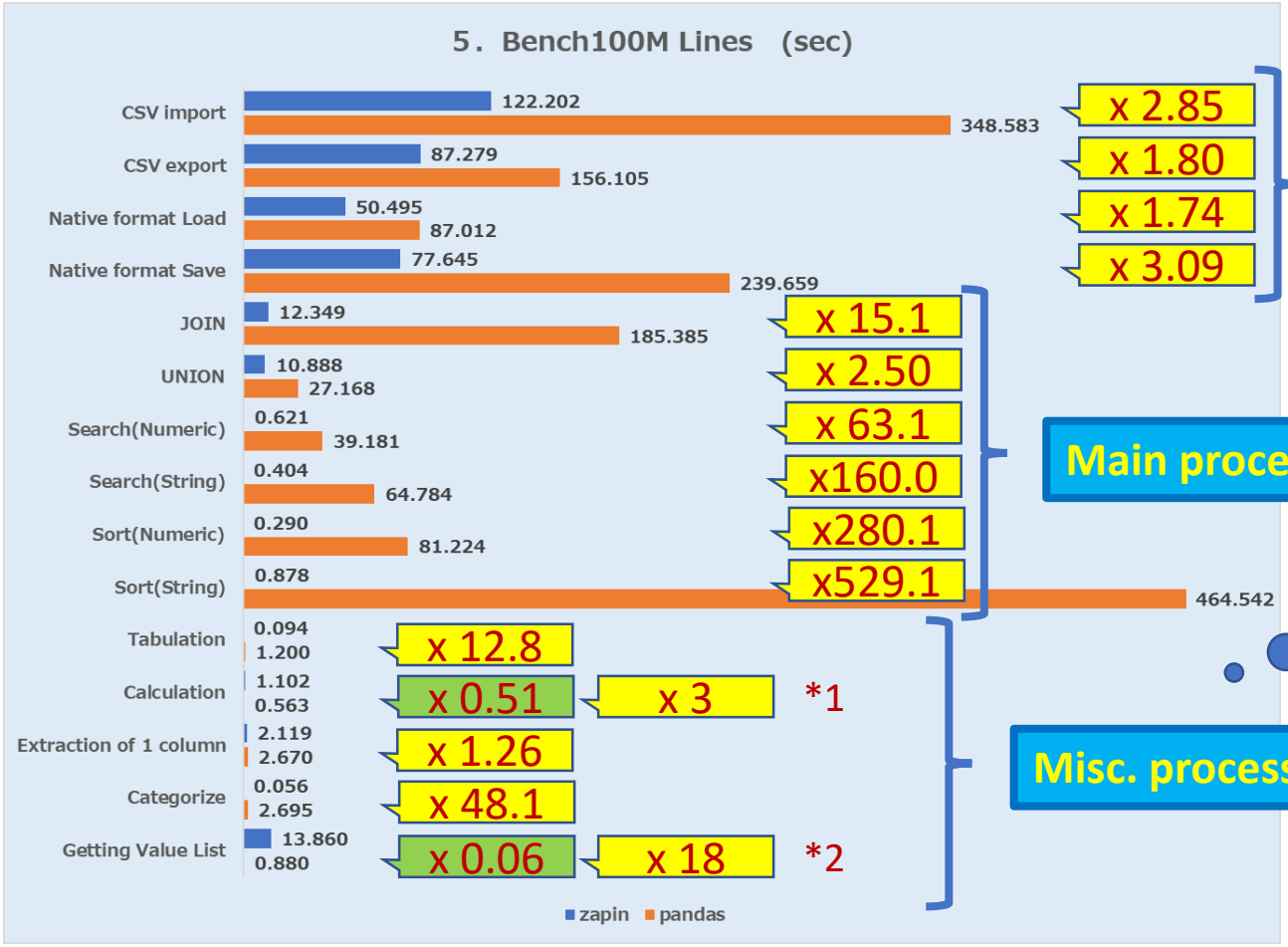
Salt Concentration (%)	Transmittance (%T)				
	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5
0	77.23	74.50	64.88	75.27	54.66
3	85.23	92.82	78.91	60.71	57.96
6	88.39	100.05	73.66	66.51	64.54
9	80.71	100.05	68.29	64.91	52.96
12	82.66	117.18	71.01	56.91	46.95
15	72.55	115.40	65.72	66.03	55.38

Zap-In's case
(every field has acceleration mechanism)

Table 1: Salt Concentration and Light Transmittance

Salt Concentration (%)	Transmittance (%T)				
	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5
0	77.23	74.50	64.88	75.27	54.66
3	85.23	92.82	78.91	60.71	57.96
6	88.39	100.05	73.66	66.51	64.54
9	80.71	100.05	68.29	64.91	52.96
12	82.66	117.18	71.01	56.91	46.95
15	72.55	115.40	65.72	66.03	55.38

Benchmark against Pandas



*1. Inadequate implementation of the calculation algorithm caused this. We will fix it expecting to raise the performance x3 than the Pandas.
*2. Miss-selection and combination of APIs caused this. If the measurement had done choosing the correct API, x18 times faster than Pandas.

Zap-In: The Elements of the Data Structure

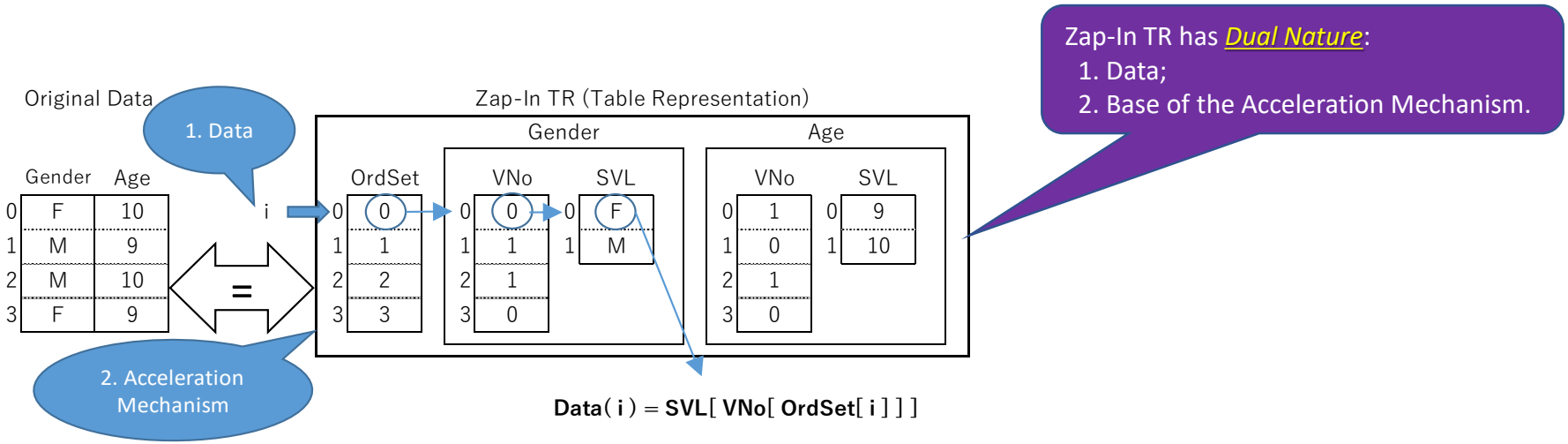
The Essential Elements of the Data Structures

Zap-In's Data Structure: Zap-In TR (Table Representation) has some elements.
The essential ones are '***Ordered Set***' and '***Sorted Value List.***'

The ***Dual Nature*** of the Data Structures

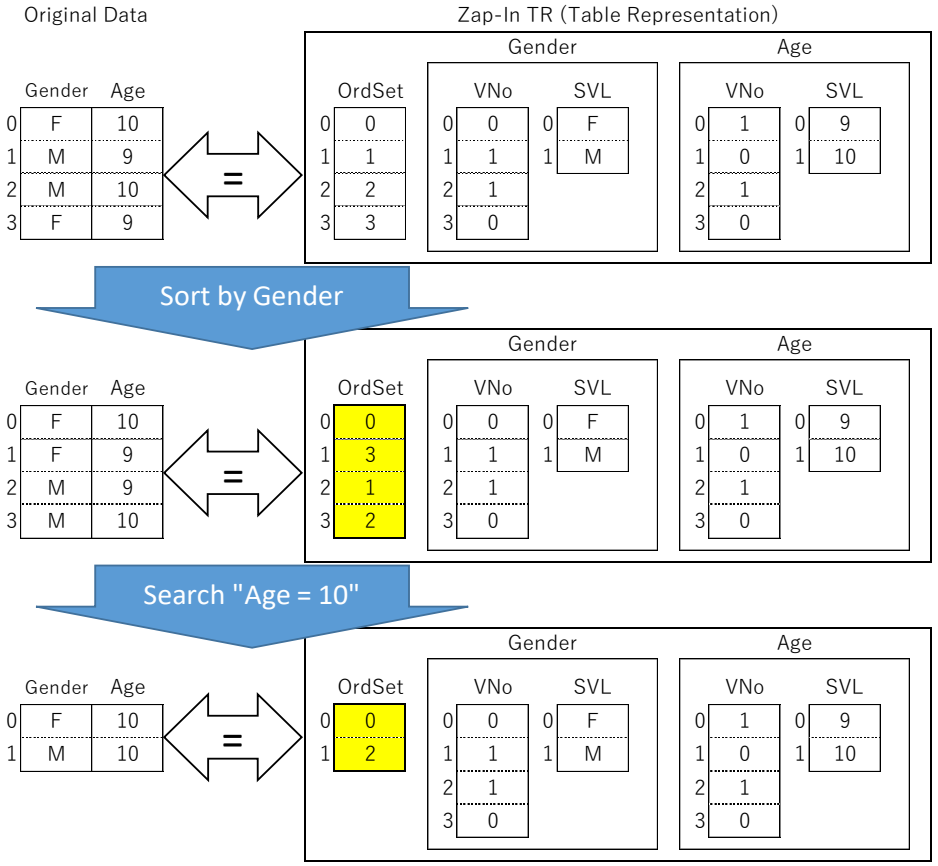
The figure below shows Zap-In TR, the data structure of Zap-In.
If you access it from the OrdSet side, like "SVL[VNo[OrdSet[i]]]," that is equivalent to the Original Data.
If you access it from other ways, it becomes an Acceleration Mechanism.

With this dual nature yields many favorable features of algorithms. I discuss it on the next page.



Zap-In: Two conditions arbitrary cascading of algorithms meaningful

Zap-In can cascade every algorithm
because its inputs/outputs are always Zap-In TR.
Moreover, the stability of sort and others makes cascades meaningful.



Zap-In **can cascade every algorithm** because every algorithm takes Zap-In TR as inputs/outputs.

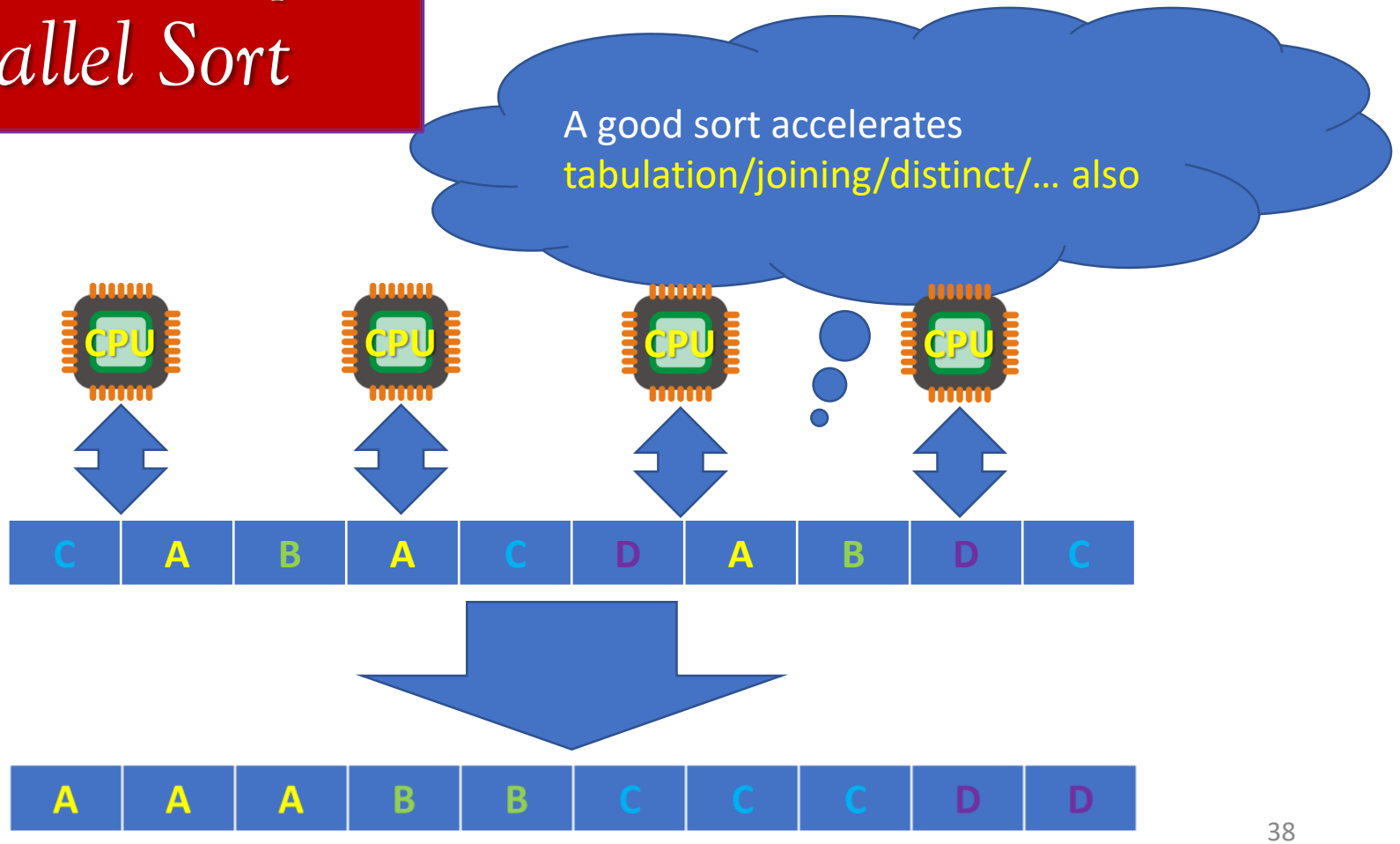
Moreover, the stability of sort and others Make cascades meaningful.

Then, **we can accelerate every combination of operations**, that was impossible when we used index technologies.

Then Zap-In can create the Big Data's spreadsheet with relational algebra support.

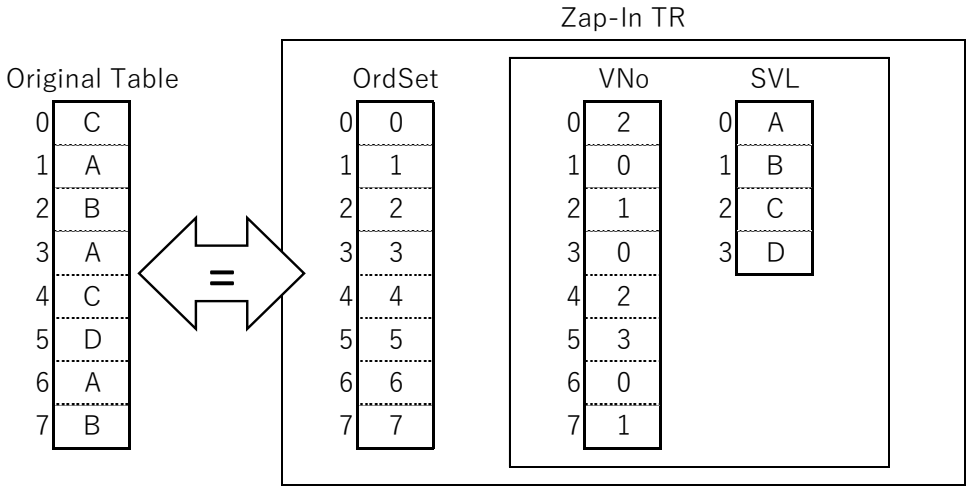
Zap-In's algorithm example: Parallel Sort

An algorithm example:
Parallel Sort



Zap-In's algorithm example: Parallel Sort

An algorithm example: Parallel Sort – 0 Initial State



Zap-In's sorting algorithm has the following features.

1. Can use multi-core easily mainly because the OrdSet is dividable.
2. Can be applied to any subset because the OrdSet is the subset.
3. Cascade-ability enabled by stability in sorting.
   Then we can resolve multi-field sort of field-A and field-B,
   into two sorts: sorting field-B at first and sorting field-A at last.

Confirm them by the explanation in pages after.

Zap-In's algorithm example: Parallel Sort

An algorithm example: Parallel Sort – 1 Count up phase

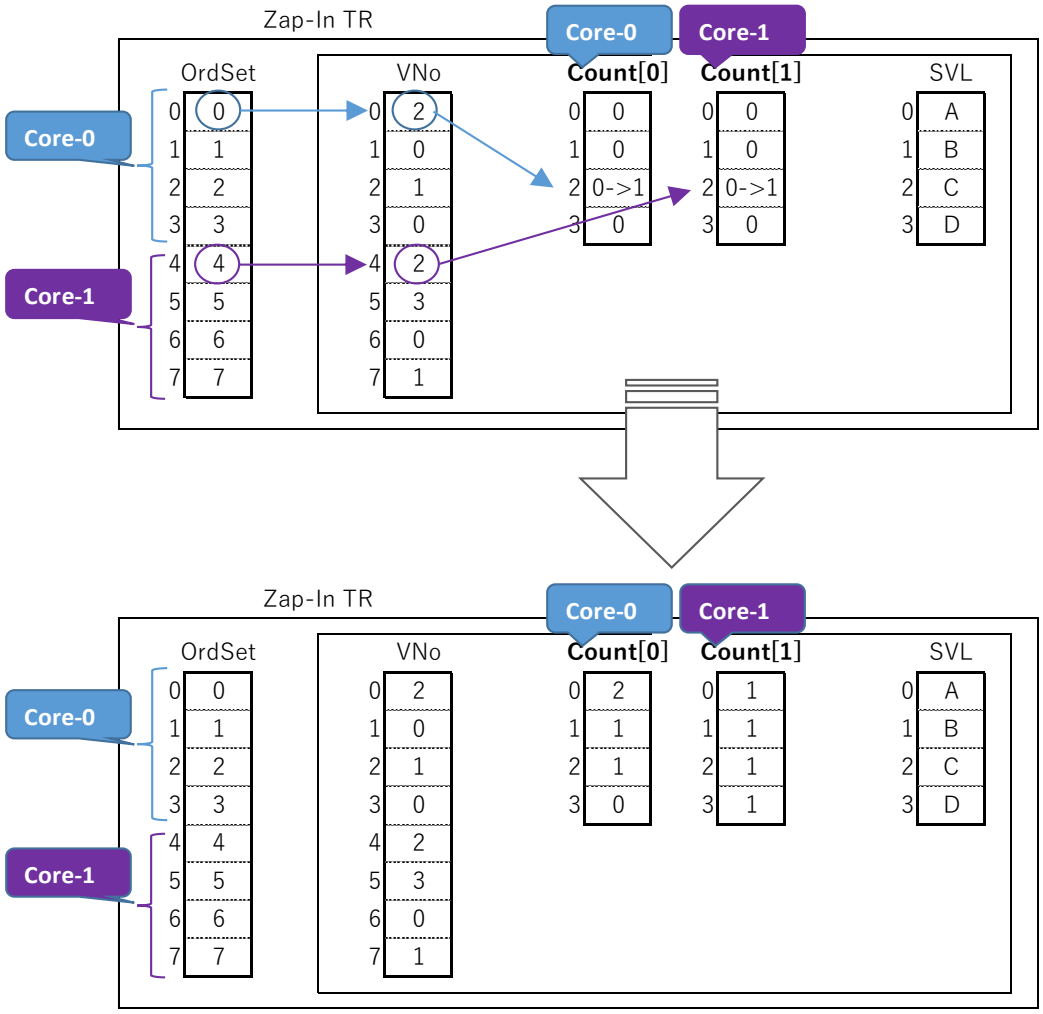
The first step of the sorting is counting up.

(Not limited to sorting, to divide OrdSet is easy. Then using multi-core becomes easy.)

At this phase, each core counts up, to know how many time each VNo value appears within each core's range in OrdSet.

```
Core-0:
for (i = 0; i <= 3; i ++)
        Count[0][ VNo[ OrdSet[i] ] ] ++;
```

Confirm that this phase is done entirely in parallel.



Zap-In's algorithm example: Parallel Sort

An algorithm example: Parallel Sort – 2 Aggregation phase

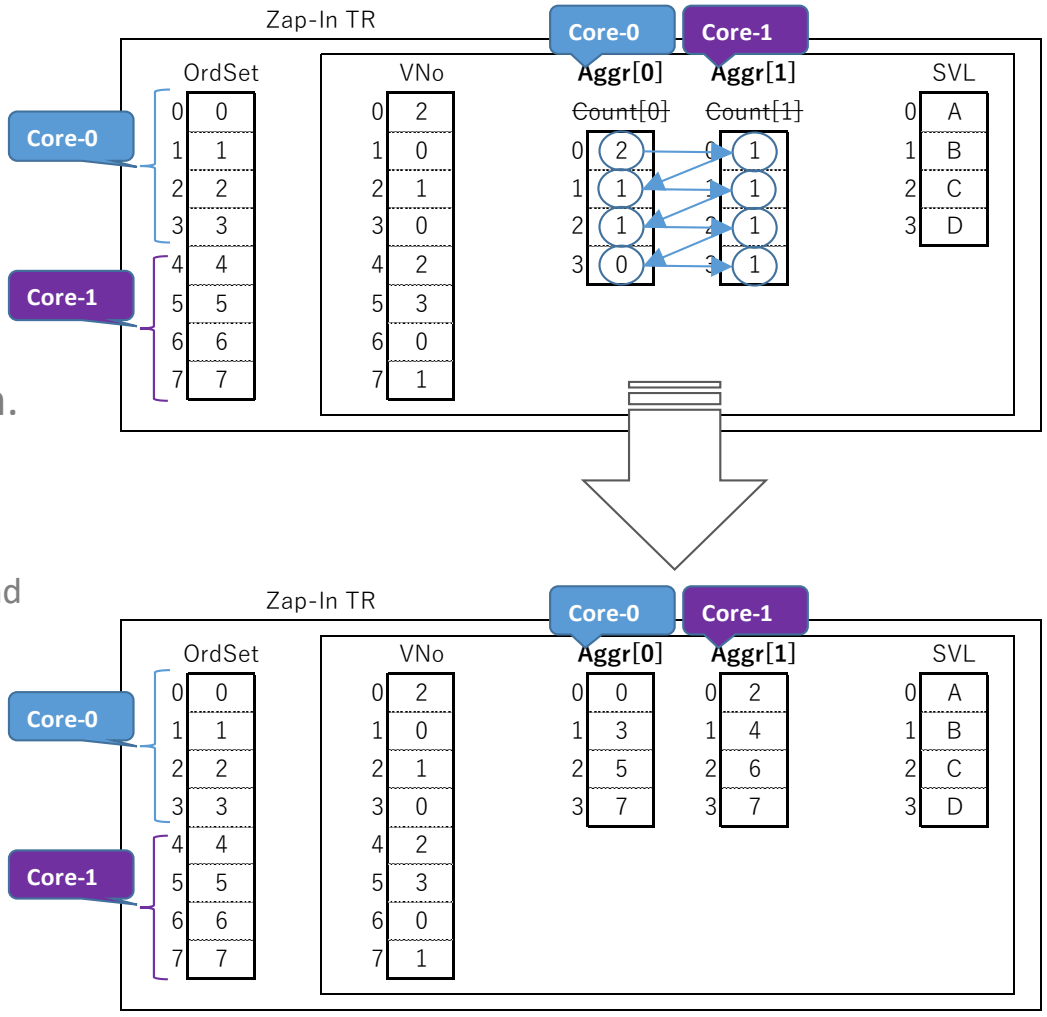
The 2nd step of the sorting is aggregation.

By aggregation, we can calculate out each group's start position in the after sort OrdSet.

That group is defined primarily by position in SVL and secondarily by core number.

```
k = 0;
for (row = 0; row <= 3; row++)
        for (col = 0; col <= 1; col++)  {
                m = Aggr[col][row];
                Aggr[col][row] = k;
                k += m;
        }
```

See, why the stability in sort is kept.



Zap-In's algorithm example: Parallel Sort

An algorithm example: Parallel Sort – 3 Transfer phase

The 3rd(final) step of sorting is transferring from the old to the new OrdSet.

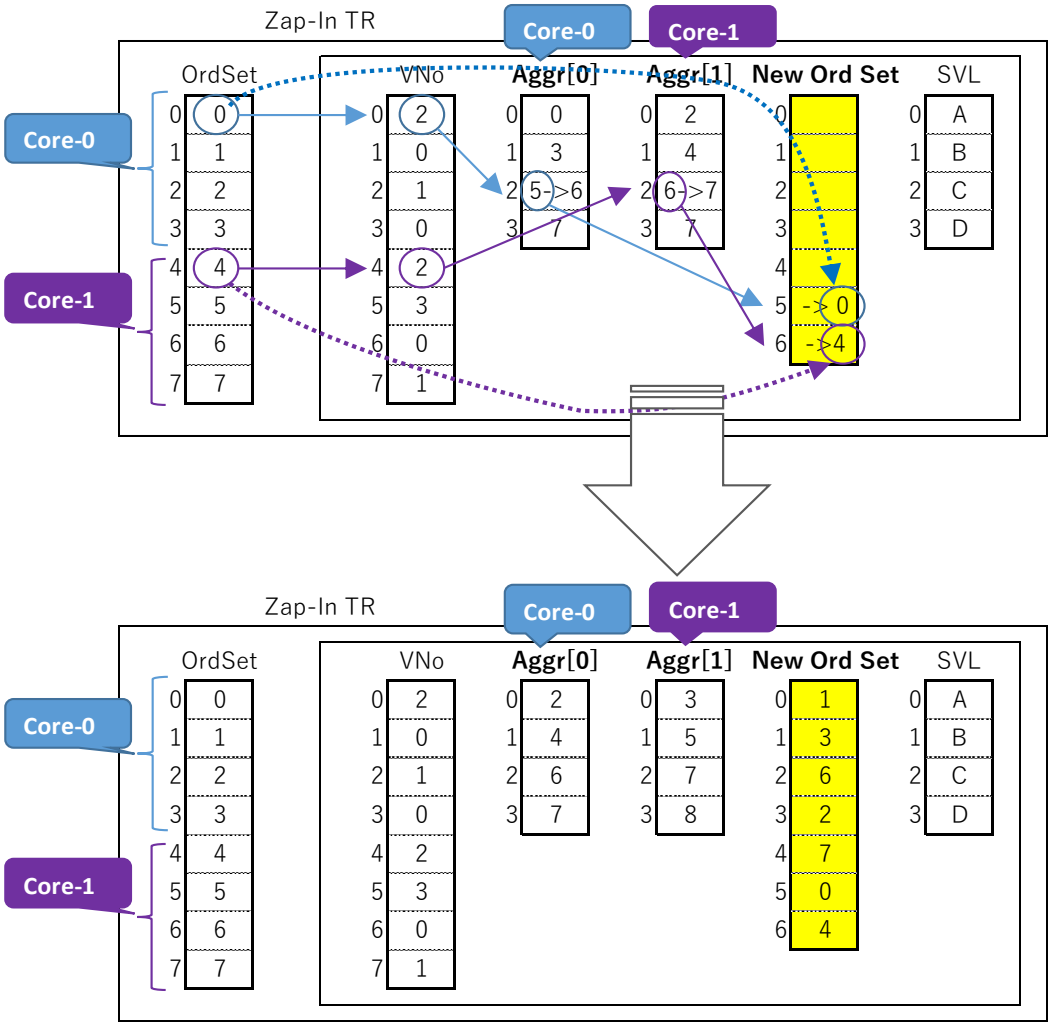
Core-0:

```
for (i = 0; i <= 3; i++) {
        k = Aggr[0][ VNo[ OrdSet[i] ] ] ++;
        NewOrdSet[k] = OrdSet[i];
}
```

Core-1:

```
for (i = 4; i <= 7; i++) {
        k = Aggr[1][ VNo[ OrdSet[i] ] ] ++;
        NewOrdSet[k] = OrdSet[i];
}
```

Confirm that this phase is done entirely in parallel.



Make The Global Utilization Cycle of Big Data;

Zap-In; What The Big Data's Spreadsheet with Relational Algebra support, achieved

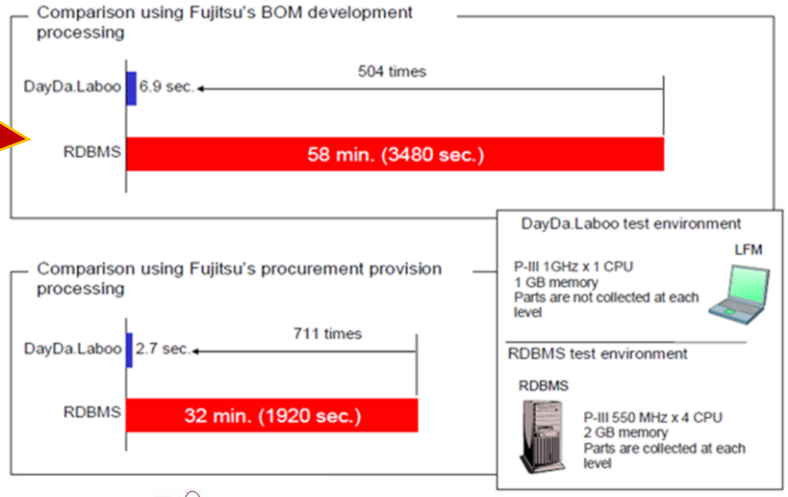
It has been working since 2002 at

Space, Nuclear, Financial, Manufacturing, Medical, Marketing, Telecommunication and Other Fields more than 200 sites.

Space

As Multi-Functional Big Data's DW

As Huge Batch Processing System, for a Computer Manufacturer



Data Import

Zap-In
- Format conversion
- Data management
- Data Processing

Necessary data, in necessary format

Nuclear

Zap-In TR
(Table Representation)

at university

Features :
- Ultra fast in handling huge data (Data Mart is unnecessary)
- Ultra fast in complicated processing (Flexible and Rich in functions)

Chemical Industory

Financial

Medical Research

Zap-In works with Zap-Over

fig. 1. Zap-In for Zap-Over

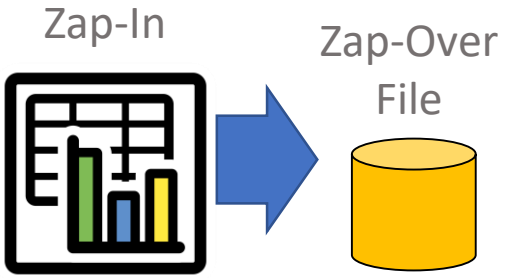
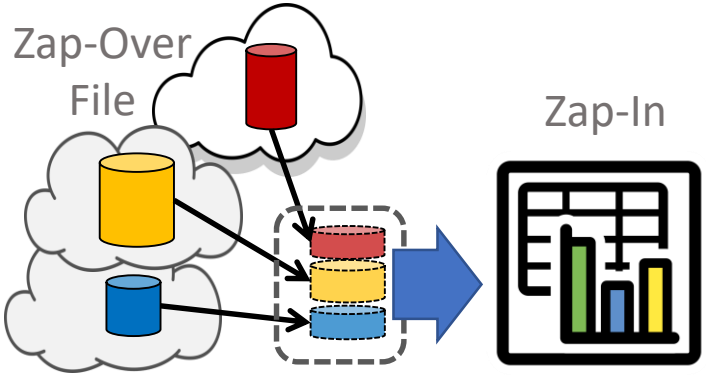


fig. 2. Zap-Over for Zap-In



Zap-In and Zap-Over works together

Zap-In for Zap-Over:

It Makes Zap-Over's Big Data File → fig 1

Zap-Over for Zap-In:

It Gathers Big Data across clouds for Zap-In → fig 2

Zap-In's Patents in U.S.A

Patents

Category	Innere code	Patent #
Searching/Tabulation/Sorting	A1-6US	US RE41,901 E1
JOINING	B1-2US	US6,721,751 B1
OLTP	B2-2US	US6,973,467 B1
Building Data Structure	B4-2US	US7,225,198 B2
Improved Joining	B5-2US	US7,184,996 B2
Joins to tree	B6-2US	US7,467,130 B2
Scope	B7-2US	US7,882,114 B2
Parallel Sort	L1-2US	US7,801,903 B2
Parallel Sort 2	L1-2DUS	US8,065,337 B2
Parallel Merge/Matching	L2-2US	US7,890,705 B2

These patents have been licensed to SAP, NEC, Fujitsu, etc.

Make The Global Utilization Cycle of Big Data;

Zap-Mass: Big Data's Massive Parallel RDB with Preemptive Multi-Tasking support

Zap-Mass: Zap Massive Parallel

Zap-Mass: Why Preemptive Multi-Tasking?

- *With Preemptive Multitasking for Massive Parallel, We Have*

1. Reliability: We can detect and swap defect nodes easily and immediately,
2. Efficiency: We can fill tasks to the idle nodes,
3. Easiness to use: We can start a task at any time,
   can watch tasks' status easily,
   can change tasks' status easily: priority, resources' mapping, others,
   can stop tasks immediately.

Make The Global Utilization Cycle of Big Data;

Zap-Mass: How Can We Make Preemptive Multi-Tasking

We have to invent "the Application + the Architecture + the Algorithms" at one time.

1. The *application* is Big Data's RDB.
2. The *inside-node architecture* should be any architecture which can deploy Zap-In.
3. The *inter-node architecture* should be like following.
   1. ***Symmetry***: Symmetric architecture makes system management easy.
   2. ***Two directions of communication paths***:
      To run many kinds of tasks at one time, we need to assign each task resources and priorities.
      We hope we can mix "high performance but expensive" and "low performance but cheap" at various levels.
      The vertical direction is high performance, and ring-wise direction is low cost.
   3. ***Divisibility***: We have to handle middle or small data sometimes. So, the architecture should be divisible.

→ *Architecture named "**The Perfect Cluster's Ring**" satisfies these features above. (See fig.1 left below)*

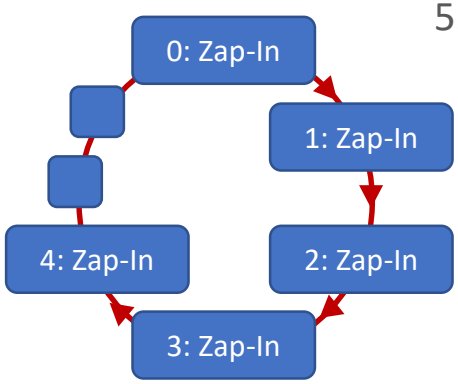


Fig. 1. The Perfect Cluster's Ring

4. The *inside-node algorithms* we use is Zap-In.
5. The *inter-node algorithms* we need should be like following.
   1. ***Universality***: Algorithms for it should cover enough kinds of processing.
   2. ***Symmetry***: Symmetric algorithms between nodes help to manage multi-tasking.
   3. ***Independence from processing order***: This also helps to manage multi-tasking.
   4. ***Divisibility of communication packets***:
      This feature helps to reduce the granularity of tasks because the tasks of each node cannot be switched while it is receiving a packet.*A

→ *Algorithms named "**Global L Operations**" satisfy these features above.*

*A. About a few microseconds, the architecture and algorithms allow different tasks to run in different nodes. Moreover, different segments(node groups) can deploy different tasks at the same time.

Zap-Mass: Architecture (The Perfect Clusters' Ring)

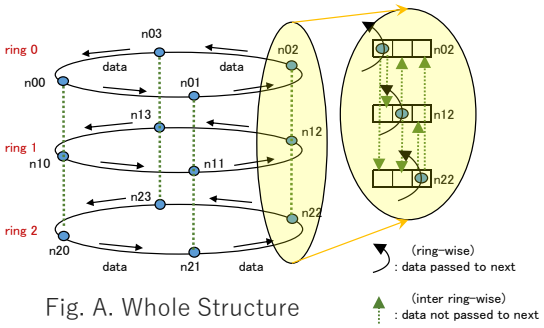


Fig. A. Whole Structure

What is "The Perfect Clusters' Ring"? (See Fig. A)

Every node belonging to one vertical group like (n02, n12, n22) has a direct path to send and receive packets to each other.
I named this architecture "Perfect Connection." It has the best performance but expensive when the number of nodes becomes huge.

On the other hand, a ring-wise path is one directional. It has low performance but low cost.

The perfect clusters' ring includes the combination of "perfect connection" and "rig-wise path," which is suitable to keep tasks having various size of data/priorities with balancing performance and cost.

(*Reprint*)

The *inter-node architecture* should be like following.

1. ***Symmetry***: Symmetric architecture makes system management easy.
2. ***Two directions of communication paths***:
   To run many kinds of tasks at one time, we need to assign each task resources and priorities.
   We hope we can mix "high performance but expensive" and "low performance but cheap" at various levels.
   The vertical direction is high performance, and ring-wise direction is low cost.
3. ***Divisibility***: We have to handle middle or small data sometimes. So, the architecture should be divisible.

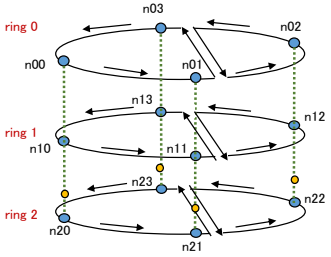


Fig. B. Division by Ring-wise

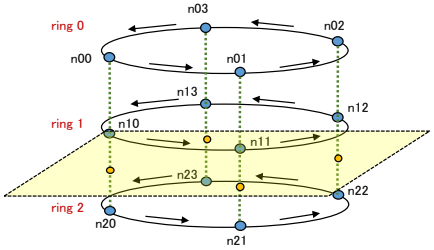


Fig. C. Division by Layer-wise

Divisibility in two directions. (See Fig. B, C)

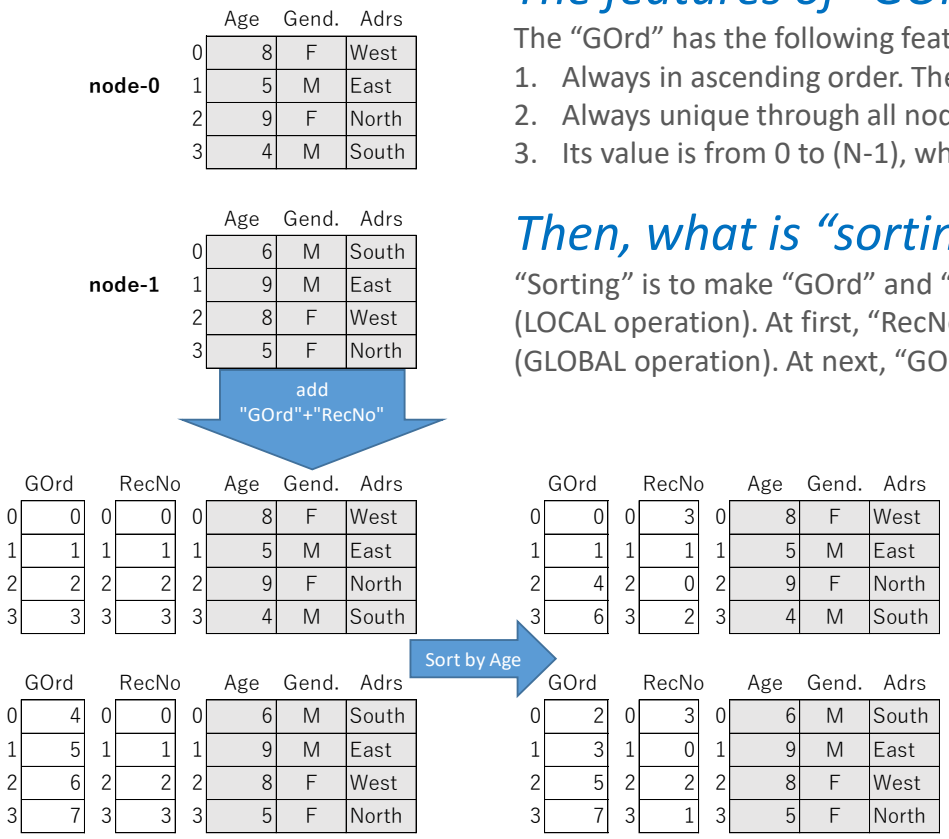
It is evident that the perfect clusters' ring is divisible in ring-wise and layer-wise. Moreover, each divided part can be divided again.
It still keeps the following three features:

1. Symmetry,
2. Two Directions,
3. Divisibility again after division.

Zap-Mass: Algorithms (Global L-Operation) (1/5)

How shall we give addresses to each record within many servers?

When we sort/search/etc. to the table, we change its record order. Swapping records takes vast cost because it moves records.
Then another option: swapping record number array is useful for a single computer system ("RecNo" array in the figure below).
However, that is not possible to the case of records are distributed to many servers.
Here I propose to separate that "RecNo" into two parts: 1st is record number in local ("RecNo"), 2nd is record order in global ("GOrd").



The features of "GOrd".

The "GOrd" has the following features.

1. Always in ascending order. Then searching out a designated value is easy.
2. Always unique through all nodes.
3. Its value is from 0 to (N-1), where N is the total count of records through all nodes.

Then, what is "sorting"?

"Sorting" is to make "GOrd" and "RecNo" in each node.
(LOCAL operation). At first, "RecNo" should be built in each node.
(GLOBAL operation). At next, "GOrd" should be built across nodes.

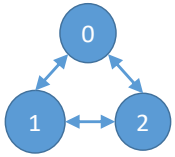
Then, what is "Global L Operation"?

Every local operation is done by Zap-In efficiently.
Then the global operation has limited variations only because that handles about the inter-node matters only.
(Almost all part has done at the local operation, at first.)

"Global L(Ladder) Operation" is the name of those global operations.

Zap-Mass: Algorithms (Global L-Operation) (2/5)

Initial State



node 0

	GOrd	RecNo		Gender	Age
0	0	0	0	F	8
1	1	1	1	M	6
2	2	2	2	M	6
3	3	3	3	F	8

node 1

	GOrd	RecNo		Gender	Age
0	4	0	0	M	7
1	5	1	1	M	6
2	6	2	2	F	7
3	7	3	3	M	8

node 2

	GOrd	RecNo		Gender	Age
0	8	0	0	F	8
1	9	1	1	F	7
2	10	2	2	F	7
3	11	3	3	M	8

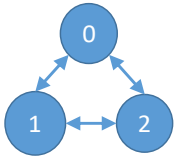
Original Image

	Gender	Age
0	F	8
1	M	6
2	M	6
3	F	8
4	M	7
5	M	6
6	F	7
7	M	8
8	F	8
9	F	7
10	F	7
11	M	8

The case of the sort by "Gender".

I explain the sort by "Gender" that starts from the initial state.
Original Image of the table in the top figure is divided into three parts and put to node 0, 1, 2.
Then initially, "GOrd" and "RecNo" should set as shown in the figure.

1st Step



node 0

	GOrd	RecNo		Gender	Age
0	-	0	0	F	8
1	-	3	1	M	6
2	-	1	2	M	6
3	-	2	3	F	8

node 1

	GOrd	RecNo		Gender	Age
0	-	2	0	M	7
1	-	0	1	M	6
2	-	1	2	F	7
3	-	3	3	M	8

node 2

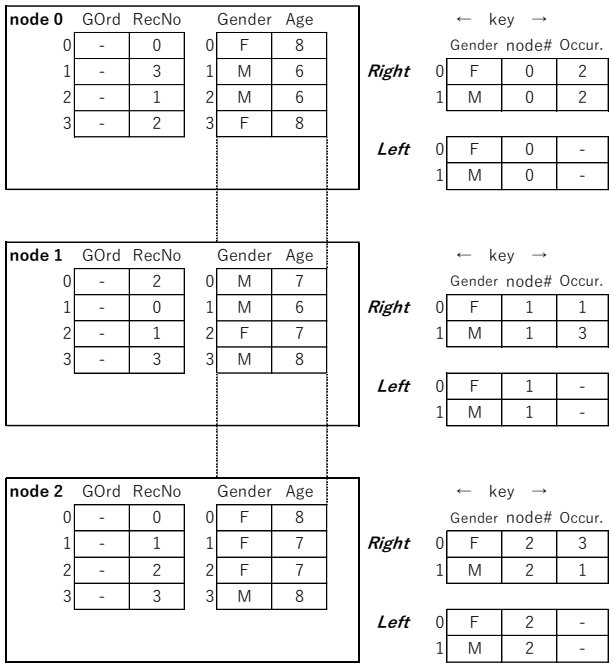
	GOrd	RecNo		Gender	Age
0	-	0	0	F	8
1	-	1	1	F	7
2	-	2	2	F	7
3	-	3	3	M	8

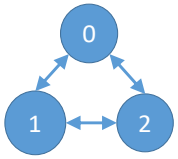
The 1st step: Local sort

The first step starts from Local sort.
You can see the result in the figure "1st Step."
At this moment "GOrd" is cleared.

Zap-Mass: Algorithms (Global L-Operation) (3/5)

2nd Step





The 2nd step: Tabulation in each node

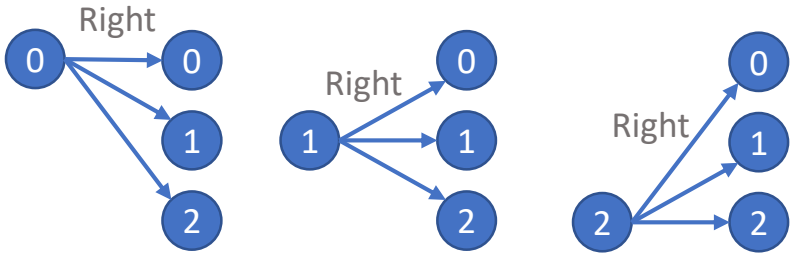
The second step is tabulation in each node.
As shown in the left upper figure, each node makes the structure shown in the figure. (I named these structures as “Right/Left L-Structure”).

The L-Structures has “key field” and “occurrence” field.
The “key field” includes “sort key’s value” and “node number.”
The “key field” is in ascending order always.

The Right L-Structure’s Occurrence field keeps occurrence count in its node.
The Left L-Structure’s Occurrence field is cleared at this moment.

Until this step, everything has done in each node. (= Local Operation)

3rd Step

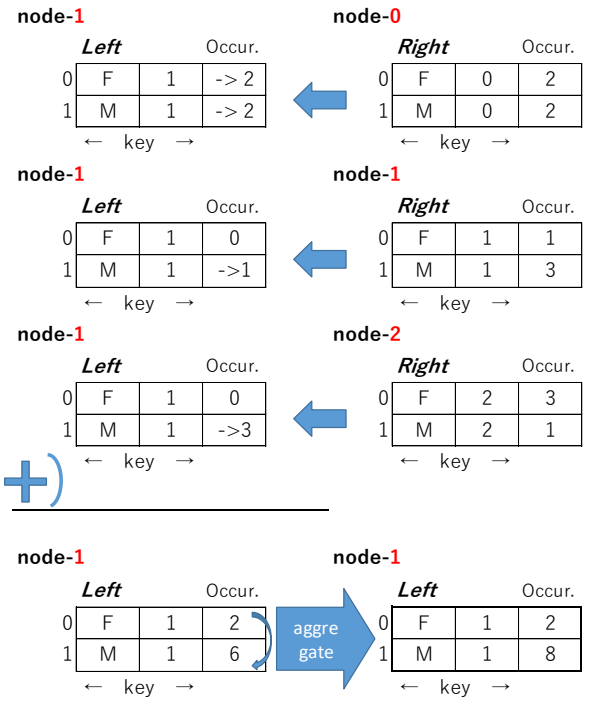


The 3rd step: Send out of “Right L-Structures”

From this third step, the Global Operation begins.
Each node sends out its Right L-Structure to every node including itself.

Zap-Mass: Algorithms (Global L-Operation) (4/5)

4th Step



node-0

	Left		Occur.
0	F	0	0
1	M	0	6

node-1

	Left		Occur.
0	F	1	2
1	M	1	8

node-2

	Left		Occur.
0	F	2	3
1	M	2	11

The 4th step: L-Operation (explained in node-1's case)

The 4th step: L-Operation has 3 sub-steps.
(note that node-1 has 3 copies of Left L-Structures at most.)
(This number: 3 is perfect cluster's size.)

Sub-step 1:

Receive Right L-Structure from node-0, node-1, node-2.
Each Left L-Structure side works as follows.

1. Pick Right key from top to bottom one by one and find out the Left key which is minimum and exceeds the Right Key.
2. Add Right L-Structure's Occurrence to Left L-Structure's found out record's Occurrence.
3. Wait every Left L-Structure updated.

Sub-step 2:

Sum up every Left L-Structure into one.

Sub-step 3:

Aggregate Left L-Structure's Occurrence.

This Left L-Structure's Occurrence means:

"GOrd" in node-0 having value: "F" starts from 0,
"GOrd" in node-0 having value: "M" starts from 6.
"GOrd" in node-1 having value: "F" starts from 2,
"GOrd" in node-1 having value: "M" starts from 8.
"GOrd" in node-2 having value: "F" starts from 3,
"GOrd" in node-2 having value: "M" starts from 11.

Note:
L-Operation efficiently runs because it is a comparison between two ascending order keys.
L-Operation succeeds independently from the order of receiving Right L-Structures.

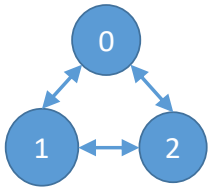
Zap-Mass: Algorithms (Global L-Operation) (5/5)

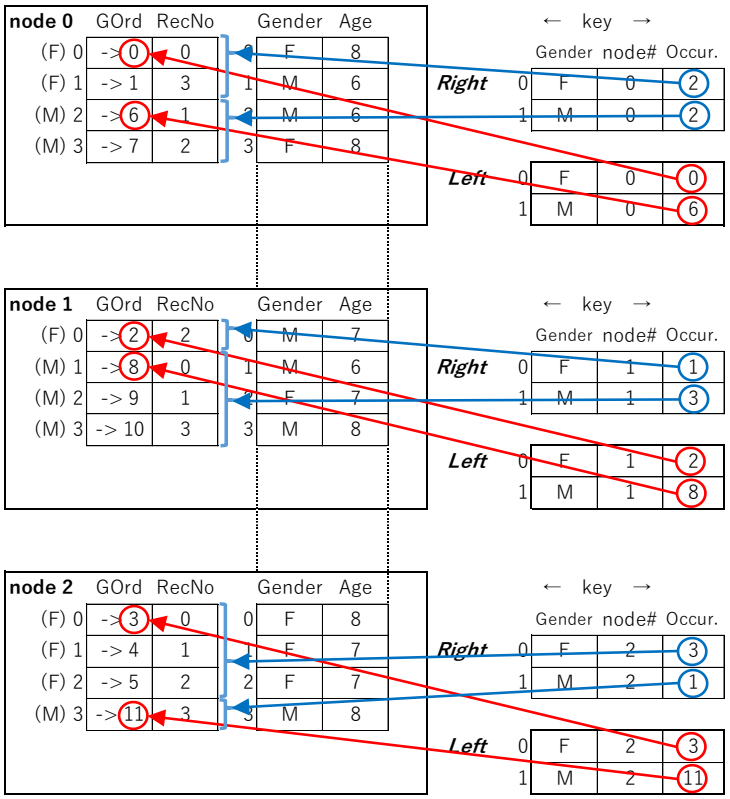
The 5th step: End up of Global L-Operation.

Each node has Right L-Structure which keeps how many times each “Gender” value exists. (In the Figure, blue arrow)

Each node has Left L-Structure which keeps GOrd’s start value corresponding to each “Gender” value. (In the Figure, red arrow)

5th Step



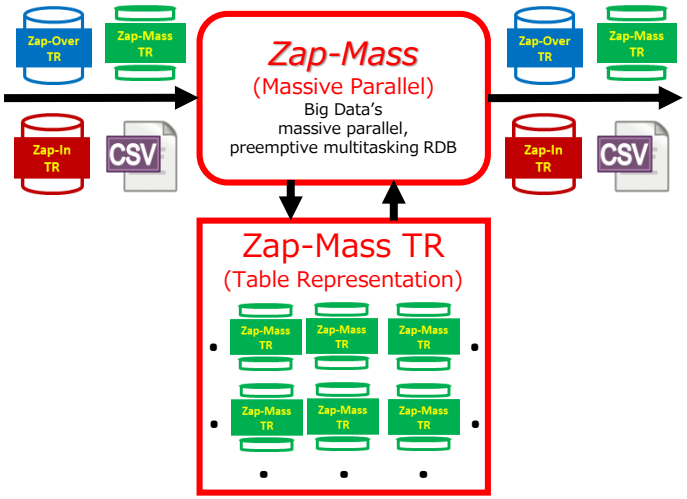


Global L-Operation satisfy following.

1. *Universality* (not proven in this document)
2. *Symmetry* (obvious by this document)
3. *Independence from processing order* (obvious by this document)
4. *Divisibility of communication packets* (not proven in this document)

Global L-Operation Contributes to Making Preemptive Multi-Tasking of Massive Parallel RDB System.

Zap-Mass: The Expected Usage



The expected usage of Zap-Mass

(Its functions are similar to Zap-In except for the size of Big Data: many trillions vs. 2 billion)

1. It imports Zap-In/Over/Mass TR or CSV.
2. It does checking/transformation/calculation/matching/… to the imported.
   If necessary, it uses local DW.
   If necessary, it saves these results in its local DW.
3. It exports process results or extracted data from its local DW, in the format of Zap-In/Over/Mass TR or CSV.

Zap-Mass: Other Issues

I mentioned about the Application / Architecture / Algorithms of Zap-Mass.
However, to design a real preemptive multi-tasking massive parallel RDB system, many other issues should be discussed.

For examples,

1. Other Global L-Operations.
2. How to switch these tasks.
3. How to check the results.
4. How to re-map resources.
5. How to manage the system.
6. How to recover from malfunctions.
7. Expected performances of Zap-Mass.

…

I make the chances to discuss them.

Make The Global Utilization Cycle of Big Data;

Zap-Mass: Patents

Several Patents are Pending

Visit our home pages to see more!

Turbo Data Laboratories, Inc.

http://turbo-data.co.jp/en/

Esperant system

https://www.ess-g.com/