White Paper

Math and the Multi-Component Keys

Gain performance benefits through multi-component keys

Author David Bayliss, Chief Data Scientist

May 20, 2013

LexisNexis®

Risk Solutions
HPCC Systems

One of the sad features of computer science is that some of the most breathtaking results are derived from the dullest of subjects. A good example of this dichotomy is the multi-component key; it is a feature that appears so insignificant that it often drops off of the marketing sheets and yet it is so fundamental that it can drive multiple orders of magnitude performance improvement. The danger presented by the unglamorous nature of multi-component keys is that they will be undervalued and thus under-utilized; and huge performance benefits may be left upon the table. The aim of this brief paper is to explain the murky math behind the phenomenal performance of the multi-component key.

Whilst multi-component keys can be utilized in many aspects of data processing for concreteness and simplicity, this paper will focus upon their use in a data search application. This is not as restrictive as it appears; most, if not all, of the more sophisticated analytic algorithms have as a pre-requisite the ability to find the data that they are going to execute upon.

The mathematics undergirding this paper is applicable to a wide range of data-files and to most existing computer systems. However, for clarity, the vehicle file discussed in 'Models for Big Data' will be used. Further, it will be assumed that the search system is a cluster where the searched for data is distributed across the cluster. The alternative to the multi-component key that is considered will be the 'key-value' pair approached championed by delivery systems from the Hadoop ecosystem. The task to be considered will be 'return all Blue, Toyotas built in 2008'.

In order for the math to have 'real numbers' rather than painful looking algebraic equations we shall list some numeric assumptions about the speed of the computers & the distribution of the data. These numbers are rounded – their purpose is to illustrate the method.

Firstly, with regard to the distributed computer system there are a number of key metrics to be measured. These are the speed that data can be transmitted over the network (100M bytes/second[1]), the speed data can be read off of disk (150M/second), the number of disk-hits that can be supported by a disk per second (200 / second) and the number of disks in a server (2). Finally, one must also allow for the scaling of the system – we shall assume 10 servers.

Secondly, one must make some assumptions about the data. We shall assume that a vehicle record is 400 bytes wide and that we have 250M of them[2]. We shall also assume that 20% of all vehicles are blue, 20% are built by Toyota and that 10% of the vehicles in our data-base were built in 2008. We shall also assume that all vehicles can be identified by a 17 byte VIN number.

In a key-value system we now have a number of alternatives for our data model. Perhaps the most natural is we make the VIN number the key and store the remaining 380 bytes in the payload. Now to perform the search for blue 2008 Toyotas, we have to iterate through our VIN numbers.

The most fatal method would be to generate all the VIN numbers and fetch them one by one. We have 10 nodes each of which can perform 400[3] fetches per second giving a total of 4000 fetches per second. That would take more than 17 hours[4].

A far better approach (assuming the system supports it) would be to have each node read and spool all the VIN data it has. Each node can read 300M/second, so ten nodes can read 3G per second allowing the data to be read in 33 seconds. A bigger problem is getting that data to the master; it can only receive 100M/second so it takes 1000 seconds but that is 17 minutes – a substantial advance on 17 hours.

---

[1] Assuming gigabit non-blocking links

[2] This makes a 100G dataset; fairly modest for a 10 node cluster.

[3] 200 fetches per second on each of 2 disks

[4] Disk caching and spindle priority elevation would ease this pain somewhat; but starting with a 17 hour process and aiming to optimize is a bad approach

In fact, the ECL[5] system has a further advance for those using the key value model – it is able to transmit the code from the master to be executed upon the slave. Thus, whilst each slave has to read in all the data – it only transmits the data for the blue, 2008 Toyotas. Using our assumptions, 0.4% (20%x20%x10%), of the data would be transmitted. Thus, the query takes 33 seconds to read and then 4 seconds to transmit. Assuming the system is smart enough to overlap those timings[6] the total cost is 33 seconds.

In fact, it is possible to approach this problem in a different way – rather than key the data solely by VIN you can key the data by the attributes with the VIN as the payload[7]. Thus, to service our example, one would create 3 keys: Make/VIN, Color/VIN, Year/VIN. For our purposes we shall assume these keys are 30, 25 and 23 bytes wide respectively. Now to service the query you do 3 fetches, one for Toyota, one for Blue and one for Year. It will be noted that the disk activity has now shrunk dramatically:

For Make of Toyota, we are going to read 50M (20% of 250M) records of 30 bytes – that is 1.5G. For Color of Blue, we read 50M records of 25 bytes (1.25G). For Year, we have read 25M records of 23 bytes (.58G). This reduces the total data read from the nodes from 100G to 3.25G – a savings of 1.5 orders of magnitude. Unfortunately, our query latency will not have come down by the same factor. In the table scan case **all** of the nodes participate in the read – in this case only those nodes servicing the part of the key with the matching values will be participating. As an absolute worst case where all the index parts happened to land on the same disk of the same node it could take 22.5 seconds to read the data. We have only saved a little time **but** we have nine nodes sitting idle able to process other queries concurrently. Of course, best case, each index read from two drives, that data could be read in 5 seconds (although fewer simultaneous enquiries can be processed).

A bigger problem is that we can no longer perform the filtering upon the nodes[8] and thus, 17 bytes from every record have to be transmitted back to the master which takes 21 seconds. We can overlap the 21 seconds with the reading – but it is 21 seconds **even if** the indexes were nicely distributed – because the master is the bottleneck. We also now only have the VIN – if we actually want ALL (or more of) the data for the vehicle we have to do fetches on each VIN returned. There are a million of them so at 4000 fetches per second we have a 4 minute (250 second) process.

From the above, assuming we only need VINs and assuming everything overlaps nicely and we get lucky with our data distribution across the nodes, we can get answers back in 21 seconds and our 10 nodes can handle 3-4 queries at once. We thus have a transaction rate of about one every 5 seconds.

As an alternative to the above, imagine rather that we had one key with the components Model/Color/Year/VIN. Based upon the previous numbers this key would be 44 bytes wide. Now to perform the query only one fetch would be required; it would probably be performed upon a single disk of a single slave and it would return 1M 44 Byte entries (44M of data). It would thus be read off of the disk in 0.3 seconds and transmitted in 0.17 seconds; most of which could be overlapped with the 0.3. We have also only maxed out one of our 20 disks and thus can perform 20 simultaneous transactions for a transaction rate of about 60 per second.

The 'VIN only' numbers are fairly dramatic, representing two orders of magnitude performance improvement. If we actually need extra vehicle data; we can use the same approach to get equally startling results. Suppose the goal is to produce a breakdown by state of the number of 2008 Blue Toyotas. In the single component key case, state data has to be stored 3 times and transmitted for every car that is Blue OR 2008 OR Toyota (approximately 50% of all vehicle) whereas in the component key, case data only has to be stored once and transmitted for vehicles which are Blue AND 2008 AND Toyota (0.4%)[9].

---

[5] http://en.wikipedia.org/wiki/ECL,_data-centric_programming_language_for_Big_Data

[6] And ECL is

[7] This is approaching the concept of an inverted index.

[8] Unless we got lucky and all the indexes happened to be upon the same node

[9] ECL is actually smart enough to perform the aggregation on the slave making data transmission negligible...

Backing away from the 'real number' example it is important to understand that the 'two orders of magnitude' is neither a lower nor upper limit upon the performance lift. The average performance lift is based upon the average cardinalities of the components in the key – and the numbers of them. Thus, for example, if you have a key with a dozen components each of which is low cardinality then a multi-component key will give four or five orders of magnitude lift. If you have two components with high cardinality the lift will be negligible.

Generalizing the concept further one must also address the question: "why not use key-value pairs but build all three columns (color, make, year) into the key?" The simple answer is that if color, make and year are the only interesting columns and if you always have all three of them – then yes you can. The far more complex answer is that to use that methodology to support a wide range of queries you would need a lot of keys.

In fact, the downside to multi-component keys is that to **simply** support a wide range of queries you need a combinatorially large number of them. Even for our simple vehicle file with 5 fields there are 10 different 3 component keys you can build. If there were 20 fields then you would need 1,140 keys; which is clearly a problem. For this reason most multi-component key systems are designed to allow one key to service multiple possible keys. Nearly all systems will allow a key of a.b.c to also function as a.b and a. Sophisticated systems such as ECL will allow the same key to service as a.c if the cardinality of b is low and b.c if the cardinality of a is low. These sophisticated mechanisms require the *system* to understand the structure of the key; the key-value systems do not allow that.

Further, it is possible to hybridize the 'single-component' and 'multi-component' approaches. Thus, a query upon five fields might be handled as two queries upon three fields which are then joined. Eg: joining a.b.c and d.e.c can give the same end result as a query upon a.b.c.d.e. It will not give the four orders of magnitude that a full five component key would give; but it does give two orders of magnitude; which is well worth having.

Much as the introduction suggested the conclusion to this paper must be bittersweet. Sweetly, a detailed understanding of an important problem can produce one or more multi-component keys that will reduce transaction times from hours[10] to sub-second. Bitterly, a performance lift of many orders of magnitude typically requires a detailed understanding of the problem. A driving principle of ECL is that the **capability is there if you need it**. If you don't need it you are always able to use a simpler and more generic model.

[10] Again it should be noted that this is not what **actually** happens. In the real world the 'many hour' problems are simply not tackled. Thus performance lifts of this magnitude really are game-changing.

For more information:

Call 877.316.9669
or visit http://hpccsystems.com

About HPCC Systems®
HPCC Systems® from LexisNexis® Risk Solutions offers a proven, data-intensive supercomputing platform designed for the enterprise to process and deliver Big Data analytical problems. As an alternative to Hadoop and mainframes, HPCC Systems offers a consistent data-centric programming language, two processing platforms and a single architecture for efficient processing. Customers, such as financial institutions, insurance carriers, insurance companies, law enforcement agencies, federal government and other enterprise-class organizations leverage the HPCC Systems technology through LexisNexis® products and services. For more information, visit http://hpccsystems.com.

About LexisNexis® Risk Solutions
LexisNexis® Risk Solutions (www.lexisnexis.com/risk/) is a leader in providing essential information that helps customers across all industries and government predict, assess and manage risk. Combining cutting-edge technology, unique data and advanced scoring analytics, we provide products and services that address evolving client needs in the risk sector while upholding the highest standards of security and privacy. LexisNexis Risk Solutions is part of Reed Elsevier, a leading publisher and information provider that serves customers in more than 100 countries with more than 30,000 employees worldwide.