

Reliable, Resilient And Efficient Distributed Storage In Dynamic Environment

Field of the Invention

This invention relates to a method and a system of increasing the reliability of a distributed storage system. More particularly, this invention relates to a method and a system of increasing the reliability of a distributed storage system by adding redundancy based on certain Quality of Service parameters.

Prior Art

Cloud computing has attracted tremendous attention from both industrial and academic societies. The shift to cloud platform enables the end-user to subscribe services from data and software solely residing on the remote data centres. In order to provide enterprise-ready services, the data stored in the remote data centres should be extremely reliable. Unfortunately, the computers in the data centres are prone to hardware malfunction. Furthermore, the computers may become unavailable for various reasons such as network congestion, power disruption and software problems. These may cause the data stored remotely to be temporarily unavailable or even permanent loss. Therefore, some systems such as Google File system or Hadoop file system create redundancy by replicating the stored data multiple times so that in the event that one copy of the data is unavailable or lost, it can still be retrieved by the replicated copies. However, such replication of files increases spatial overhead. For example, if a system implements replication rate of three, the spatial overhead is around 200% of the original data. Most of the time, the default replication rates are even higher in order to achieve a satisfied availability rates. This causes extremely large spatial overhead. The large spatial overhead also introduces additional internal load within the cloud and limits the overall capacity.

Therefore, those skilled in the art are constantly striving to provide a method and a system that can provide an improved method of storing file in a distributed storage system.

Summary of the Invention

The above and other problems are solved and an advance in the art is made by the method and system in accordance with the present invention. A first advantage of a method and system in accordance with embodiments of this invention is that it achieves higher reliability with much lower spatial redundancy. A second advantage of a method and system in accordance with embodiments of this invention is that the storage system can be used more efficiently since lower spatial redundancy is achieved. A third advantage of a method and system in accordance with embodiments of this invention is that the space required for actual physical storage can be greatly reduced since the storage system is more efficiently used. A fourth advantage of a method and system in accordance with embodiments of this invention is that the power can be conserved. A fifth advantage of a method and system in accordance with embodiments of this invention is that the failure rate is reduced.

In accordance with embodiments of this invention, a method for storing file in a distributed storage network is provided. The method may be performed in the following manner. A file to be stored is read and a code rate, R , and a block size, n , is determined. The method then divides the file into k segments and generates an erasure code. Subsequently, the method encodes k segments of the file based on the erasure code and generates encoded n segments. The encoded n segments are transmitted to the distributed storage network. Preferably, $R=k/n$.

In accordance with embodiments of this invention, the method for determining the code rate, R , and block size, n , is provided in the following manner. The file size and the QoS parameters are determined. Based on the QoS parameters and the file size, a range of code rates and block sizes are determined. The method then calculates the optimum code rate and block size.

In accordance with embodiments of this invention, the method for calculating the optimum code rate and block size is provided in the following manner. The transmission overhead and I/O constraints, internal load, spatial overhead and system performance are

determined. Based on these information, the optimum code rate and block size are determined.

In accordance with embodiments of this invention, a system for storing file in a distributed storage network is provided. The system includes a processor and a memory having instructions to add redundancy to a file. The instructions to add redundancy may be performed in the following manner. A file to be stored is read and a code rate, R , and a block size, n , is determined. The method then divides the file into k segments and generates an erasure code. Subsequently, the method encodes k segments of the file based on the erasure code and generates n encoded segments. The encoded n segments are transmitted to the distributed storage network.

Brief Description of the Drawings

The above and other features and advantages of a method and/or system in accordance with this invention are described in the following Detailed Description and are shown in the following drawings:

Figure 1 illustrates an overview of a system in accordance with an embodiment of the invention;

Figure 2 illustrates an exemplary processing system that represents a processing system in a computer system used by the system in accordance with an embodiment of the invention;

Figure 3 illustrates a software module executable by the computer system in accordance with an embodiment of the invention;

Figure 4 illustrates a flow diagram of a process for adding redundancy performed by the system in accordance with an embodiment of the invention;

Figure 5 illustrates a flow diagram of a process for determining a code rate and block size performed by the system in accordance with an embodiment of the invention;

Figure 6 illustrates a flow diagram of a process for determining the optimum code rate and block size based on the system performance performed by the system in accordance with an embodiment of the invention;

Figure 7 illustrates a diagram showing the relation between spatial redundancy,

failure rates and number of chunks;

Figure 8 illustrates the results of an example of storing a file having a file size of 5MB;

Figure 9 illustrates the results of an example of storing a file having a file size of 10GB; and

Figure 10 illustrates a diagram showing the relation between failure rate and storage overhead.

Detailed Description

This invention relates to a method and a system of increasing the reliability of a distributed storage system. More particularly, this invention relates to a method and a system of increasing the reliability of a distributed storage system by adding redundancy based on certain Quality of Service parameters.

It is envisioned that a system and/or method in accordance with embodiments of this invention increases the reliability of storing of file in a distributed storage system with less spatial overhead. In particular, the system adds redundancy in an effective way so that files to be stored in a distributed storage system need not be replicated multiple times. The system is realised by first dividing the file to be stored into k segments. Each bit of the k segment is encoded to n segments by using Low Density Parity Check (LDPC) code. The parameter n and k are decided by the size of the file and the Quality of Service (QoS) parameters which includes transmission delay and availability rates. The n segments are then physically stored in different nodes in the cloud storage system or any distributed storage system. Advantageously, even if some of the nodes are down for some reasons, the system can still recover the whole file by using the decoding algorithms of the LDPC codes. One skilled in the art will recognise that although LDPC code is described above, other erasure code such as Reed Solomon may also be implemented without departing from the invention.

Figure 1 illustrates a system 100 that performs processes for providing the systems and methods in accordance with embodiments of this invention. The system 100 includes

a computer system 110 and storage systems 131-134.

Computer system 110 is a desktop computer, laptop computer, tablet, mobile device or other computer terminal that executes applications that perform the required processes in accordance with this invention. Essentially, computer system 110 receives a file to be stored. The computer system 110 will slice and encode the file based on certain QoS parameters and convert the file to a number of chunks. Further details of the processes performed by the computer system 110 will be described below with reference to figures 4-6.

Computer system 110 is communicatively connected to storage systems 131-134 via a conventional wire connection, wireless connection or any method through network 120. One skilled in the art will recognize that although only one computer system 110 is shown, any number of computer system 110 may be connected without departing from this invention.

Storage systems 131-134 are processing systems that provide resources to computer system 110 and are connected to the network 120 via a conventional wire connection, wireless connection or any other method. Resources may include but are not limited to storage, processing time, and applications. One skilled in the art will recognize that although only storage systems 131-134 are shown, any number of storage systems may be connected without departing from this invention. One skilled in the art will recognise that storage systems 131-134 may also be directly connected to the computer system 110 without departing from the invention.

Figure 2 illustrates an exemplary processing system 200 that represents the processing systems in computer system 110 that execute instructions to perform the processes described below in accordance with this invention. One skilled in the art will recognize that the instructions may be stored and/or performed as hardware, firmware, or software without departing from this invention. One skilled in the art will recognize that the exact configuration of each processing system may be different and the exact configuration executing processes in accordance with this invention may vary and processing system 200

shown in Figure 2 is provided by way of example only.

Processing system 200 includes Central Processing Unit (CPU) 205. CPU 205 is a processor, microprocessor, or any combination of processors and microprocessors that execute instructions to perform the processes in accordance with the present invention. CPU 205 connects to memory bus 210 and Input/ Output (I/O) bus 215. Memory bus 210 connects CPU 205 to memories 220 and 225 to transmit data and instructions between the memories and CPU 205. I/O bus 215 connects CPU 205 to peripheral devices to transmit data between CPU 205 and the peripheral devices. One skilled in the art will recognize that I/O bus 215 and memory bus 210 may be combined into one bus or subdivided into many other busses and the exact configuration is left to those skilled in the art.

A non-volatile memory 220, such as a Read Only Memory (ROM), is connected to memory bus 210. Non-volatile memory 220 stores instructions and data needed to operate various sub-systems of processing system 200 and to boot the system at start-up. One skilled in the art will recognize that any number of types of memory may be used to perform this function.

A volatile memory 225, such as Random Access Memory (RAM), is also connected to memory bus 210. Volatile memory 225 stores the instructions and data needed by CPU 205 to perform software instructions for processes such as the processes required for providing a system in accordance with this invention. One skilled in the art will recognize that any number of types of memory may be used as volatile memory and the exact type used is left as a design choice to those skilled in the art.

I/O device 230, keyboard 235, display 240, memory 245, network device 250 and any number of other peripheral devices connect to I/O bus 215 to exchange data with CPU 205 for use in applications being executed by CPU 205. I/O device 230 is any device that transmits and/or receives data from CPU 205. Keyboard 235 is a specific type of I/O that receives user input and transmits the input to CPU 205. Display 240 receives display data from CPU 205 and display images on a screen for a user to see. Memory 245 is a device

that transmits and receives data to and from CPU 205 for storing data to a media. Network device 250 connects CPU 205 to a network for transmission of data to and from other processing systems.

Figure 3 illustrates an exemplary software module 300 that is part of processing system 200 that executes instructions to perform the processes described below in accordance with this invention. Software module 300 includes a slicer/combiner 310, QoS control 320, encoder/decoder 330 and erasure code generator 340. File to be stored is read by the software module 300. The QoS control determines a set of parameter such as code rate and block size to be used for the slicer/combiner 310 and matrix generator 340. The set of parameter is determined by the size of the file, a QoS parameter related to the reliability, and a QoS parameter related to the latency. One skilled in the art will recognise that other QoS parameters may be implemented without departing from the invention.

Slicer/combiner 310 slices or divides the file to a determined number of file chunks. Slicer/combiner 310 may also reconstruct the original whole file by combining the sliced file chunks. Matrix generator 340 generates an erasure code for encoder/decoder 330 to encode or decode the file. Encoder/decoder 330 encodes or decodes file based on an erasure code generated by the erasure code generator 340. After encoding, encoded file chunks are generated which may be the same as or different from the original file chunks. Redundant chunks are also generated to provide enhanced reliability. Although figure 3 shows only one redundant chunk, one skilled in the art will recognise that more than one redundant chunk may be generated as long as the number of file chunk is more than the number of redundant chunk.

One skilled in the art will recognize that the exact configuration of each component in software module 300 may be different and the exact configuration executing processes in accordance with this invention may vary and software module 300 shown in Figure 3 is provided by way of example only.

Figure 4 illustrates a flow diagram of process 400 performed by a system in computer system 110 in accordance with an embodiment of this invention. The system is provided by processes stored in software module 300, firmware, or hardware. Process 400 begins in step 405 by reading a file to be stored. A code rate (k) and block size (n) is then determined in step 410. The steps of determining the code rate and block size will be described further with reference to figures 5 and 6 below. The code rate represents the proportion of the data stream that is useful. If the useful data size is k bit, n-k parity bits are attached after encoding, the corresponding code rate is $R=k/n$ and block size is n.

In an example where LDPC is applied in the storage systems 131-134, the file to be stored is firstly divided into k segments. Then for each bit of the k segments, n-k parity segments are created by LDPC encoding and distributed to the storage systems 131-134. Therefore, the spatial redundancy required for code rate R is computed as follows,

$$Sr(R, F_Size) = \frac{F_Size}{k} n = \frac{F_Size}{R}$$

Where F_Size represents the size of the file to be stored. For the forgoing, it is assumed that k is the number of segments that the file to be stored is divided into, n is the number of segments after LDPC encoding and R refers to the code rate of LDPC and $R = k/n$.

In step 415, process 400 divides the file into k segments based on the determined block size using erasure codes such as Reed Solomon code or Low Density Parity Check (LDPC) code. Process 400 then generates an erasure code in step 420. Based on the erasure code, process 400 encodes segments of the file using the matrix to generate redundant segments in step 425. If LDPC code is used, segments of the file will be encoded using the matrix to generate n-k parity segments.

In step 430, process 400 transmits the encoded redundant segments and encoded file segments to storage systems 131-134. Process 400 ends after step 430.

Figure 5 illustrates a flow diagram of process 500 performed by a system in computer system 110 in accordance with an embodiment of this invention. The system is provided by processes stored in software module 300, firmware, or hardware. Process 500 is an embodiment of a process for performing step 410 in process 400.

Process 500 begins with step 505 by determines the file size of the file to be stored. Process 500 then determines the QoS parameters. The QoS parameters are availability rates and latency.

Availability rates represent the reliability that the system provides. The following equation (1) shows how the availability rate is computed.

$$A_{ldpc}(k, n) = \sum_{i=|fk|+1}^n \binom{n}{i} \epsilon^i (1 - \epsilon)^{n-i} \quad (1)$$

This is based on the assumption that each segment of the file is stored in a single computer node and the availability rate for a single node is ϵ . As LDPC is a near-optimal code, it means that the LDPC code requires average $|fk| + 1$ number of segments in order to reconstruct the original file, where f is called as an overhead factor. It is worth mentioning if other erasure code is used, $|fk| + 1$ will be replaced by other numbers. For example, if Reed-Solomon is used, $|fk| + 1$ will be replaced by $k+1$.

Figure 7 shows the relation between the spatial redundancy, failure rates and the block size. Line 740 shows two times redundancy (i.e. 200% spatial overhead), line 730 shows one time redundancy (i.e. 100% spatial overhead), line 720 shows 0.75 times redundancy (i.e. 75% spatial overhead) and line 710 shows 0.5 times redundancy (i.e. 50% spatial overhead). It can be observed that minimum k is allowed to be smaller when the code rate is lower. This means if the rate of redundancy is higher, the number of chunks can be smaller.

Latency is a measure of time delay experienced in a system. Regarding to the latency of storing and downloading the file, it can be contributed by two major factors. The first factor is the encoding and decoding latency. LDPC encoding and decoding only involves AND/XOR operations and its complexity is increased linearly with the block size. The corresponding latency introduced by the encoding and decoding does not differentiate among each other when different LDPC parameters are applied. Therefore, the first factor will not be considered for the metrics introduced for the optimization of the parameters. The second factor is the latency caused by transmission. To minimize the transmission delay of the stored files, especially for the files with large size, parallel uploading or downloading are normally enabled. As a result, the transmission delay varies a lot if the file is divided into chunks of different sizes. The equation (2) below shows the latency introduced by the transmission.

$$T_L(k) = t_a \frac{F_Size}{k} \quad (2)$$

Where t_a represents the average transmission time/bit.

In step 515, process 500 determines the range of code rates and block sizes. An example of the determination of the range of code rates and block sizes is provided as follows.

S/N	QoS Parameter	Value
1	Availability rate	$\geq 99.9999\%$
2	Latency for an image file	≤ 5 seconds
3	Latency for a HD Video	≤ 5 minutes

Table 1

Spatial Overhead (redundancy)	Code Rate	Minimum number of chunks, k
200%	1/3	4
100%	1/2	12
75%	4/7	21
50%	2/3	58

Table 2

F-Size	Minimum number of chunks, k
5MB	1
10GB	6

Table 3

Assuming the QoS parameters are provided in table 1 above, the minimum requirement for the number of segment for different code rates determined via equation (1) is provided in table 2 above. Another requirement for the number of segment is determined via equation 2 and is provided in table 3 above. The F_Size of 5MB is for image file while the F_Size of 10GB is for HD Video file.

Process 500 then determines an optimum code rate and block size in step 520. The steps of determining the optimum code rate and block size will be described further with reference to figure 6 below. Process 500 ends after step 520.

One skilled in the art will recognise that steps 505 and 510 in process 500 need not be sequential and the order in which being performed may be swapped. Specifically, step 510 may be performed prior to step 505 without departing from the invention. Further, steps 505 and 510 may also be performed concurrently without departing from the invention.

Figure 6 illustrates a flow diagram of process 600 performed by a system in computer system 110 in accordance with an embodiment of this invention. The system is provided by processes stored in software module 300, firmware, or hardware. Process 600 is an embodiment of a process for performing step 520 in process 500.

Process 600 begins with step 605 by reading the range of code rates and block sizes. In step 610, process 600 determines the transmission overhead and I/O constraints.

Transmission overhead refers to the communication overhead caused by computer system 110 and service provider when transferring the segments of the file over the network 120. The communication overhead includes the overhead caused by the remote procedure call (RPC) when establishing the links between the computer system 110 and the storage systems 131-134. However, the transmission overhead can be ignored if it is compared with those introduced by transferring the file/data stored. Furthermore, the overhead introduced by transferring the actual file/data is usually determined by the total size of the file/data and not influenced by the LDPC configuration. Therefore, the metric for transmission overhead is usually of lower priority in the metrics when optimizing the parameters.

The I/O constraints refer to the limit of the number of connections a storage system 131-134 can handle. The corresponding metrics and relation with the number of the segments are defined in equation (3) as follows,

$$T_o(t, k) = \frac{I(t)k}{i_{Total}} \quad (3)$$

Where T_o is the average overhead increased by a single segment. A time-varying coefficient is introduced to the metrics. The time-varying coefficient introduces the time-varying weighting effect for the impact of the increase of the connection number. The impact should be increasing when fewer connections are left. This is because the resource for connection number becomes more and more precious when there is fewer left. For

example, the percentage of used I/O connection can be applied as $I(t)$. In this way, when the percentage of the usage is high, it will contribute more to the final metrics so that the configuration leading to fewer I/O connections is to be favoured.

In step 615, process 600 determines the internal load. In a Hadoop Distributed File System (HDFS), segment reports are required to be sent periodically to inform the server for the locations of the segments of the stored files. The internal load is dependent on the type of storage systems used. If HDFS is used and as HDFS has a single-node namespace server architecture to achieve linear performance scaling, the system capacity of the system is limited by the amount of RAM and the processing capability of the namespace server. The internal load includes overhead to transmit and process segment reports, metadata overhead and etc. The equation (4) below shows the internal load caused by two types of resources. The first type is caused by the memory required to store the metadata which contains the mapping between the blocks and their physical locations. The second type is the processing power required to process the block reports.

$$\begin{aligned} I_o(t, k, R) &= \frac{L(t)S_{Size}n}{L_{Total}} + \frac{M(t)n}{M_{Total}} \\ &= \left(\frac{L(t)S_{Size}}{L_{Total}} + \frac{M(t)m_o}{M_{Total}} \right) k/R \end{aligned} \quad (4)$$

Where S_{Size} represents the size for a segment report and m_o represents average internal load introduced by a single segment. Similarly to the $I(t)$ added to the metrics of I/O constraints, two time-varying coefficients are introduced to the two types of metrics.

In step 620, process 600 determines the spatial overhead. The metric for spatial overhead can be determined via equation (5) below.

$$S_o(R, F_Size) = F(t) \frac{S_r(R, F_Size) - F_Size}{S_{Total}} = F(t) \frac{F_Size(1-R)}{S_{Total}R} \quad (5)$$

Where S_{Total} represents the total storage capacity of the system and $F(t)$

represents the percentage of the storage already in use.

In step 625, process 600 determines the system performance. In order to optimize the parameters, a metrics is required to quantify the system performance in terms of various resources. A system performance matrix (SPM) is proposed as a function of all the metrics introduced above as follows.

$$SPM = f(T_o, I_o, S_o) \quad (6)$$

One skilled in the art will recognise that the equation (6) above is based on the actual system configuration. Based on the cost of each type of the resources including I/O connection, the memory, hard disk and processing power, the coefficients of each metrics can be adjusted. For example, if the I/O connection is of higher priority compared with the rest of the metrics, then the coefficient for T_o should be bigger compared with those for I_o and S_o . The equation (7) below shows the simplest configuration of SPM where T_o , I_o and S_o are combined linearly.

$$SPM(k, R) = \frac{I(t)k}{I_{Total}} + \left(\frac{L(t)S_{Size}}{L_{Total}} + \frac{M(t)m_o}{M_{Total}} \right) k/R_o + F(t) \frac{F_{Size}(1-R)}{S_{Total}R} \quad (7)$$

In terms of $I(t)$, $L(t)$, $M(t)$ and $F(t)$, it is proposed that they should represent the percentage of the resource already in-use. The selected parameters for LDPC code are proposed as those yields the minimal SPM value, which is represented by the following equation.

$$\{\hat{k}, \hat{R}\} = \min SPM(k, R) = \min f(T_o, I_o, S_o) \quad (8)$$

The number of encoded segments increases with the code rate. This means that lower code rate is expected to achieve less internal load and fewer I/O connections. Lower code rate leads to higher spatial overhead. This trade-off is well represented by SPM. Furthermore, $I(t)$, $L(t)$, $M(t)$ and $F(t)$ represent the percentage of the resource already in-use.

if the percentage of the used particular resource is higher at time t , the algorithm makes the weighting of that particular resource higher. As a result, the algorithm favours the configuration that gives fewer increments for that particular resource to ease the situation of over consumption of the resource.

Based on the system performance, process 600 determines the optimum code rate and block size in step 630. Process 600 ends after step 630.

One skilled in the art will recognise that steps 610-620 in process 600 need not be sequential and the order of steps 610-620 may be interchangeable without departing from the invention. Furthermore, steps 610-620 may also be performed concurrently without departing from the invention. One skilled in the art will also recognise that process 600 need not require all the three metrics in steps 610-620 in order to work the invention. Furthermore, any other metrics that are related to the system performance of the distributed storage system may be included without departing from the invention.

An example is provided as follows to illustrate the steps of how the parameters of LDPC code are adjusted. Hadoop is used as an example for the distributed file system. The Hadoop distributed file system (HDFS) is an open source system being used by numerous organizations including Facebook and Yahoo.

Table 4 shows the parameters and coefficients chosen for the illustration. The parameters 1 to 8 are based on the statistics from HDFS. The parameter 8 is selected as 50Mbps which is a high speed broadband network speed in Singapore. The parameter 10 represents all the time-varying coefficients and the percentage of the used resources is all assumed to be 1%.

S/N	Parameter Name	Value
1	Number of Nodes	10,000
2	Storage per node	6 TB
3	Maximum socket number per node	10,000
4	I_{Total}	(Number of nodes) x (Maximum socket number per node) = 10,000 x 10,000 = 10^8
5	L_{Total}	60GB
6	M_{Total}	(Number of nodes) x (Maximum number of blocks per node) = 10,000 x 60,000 = 6×10^8
7	S_{Total}	(Number of nodes) x (Storage per node) = 10,000 x 6×10^{12}
8	S_{Size}	200B
9	t_a	1/50Mbps
10	$I(t), L(t), M(t), F(t)$	1%

Table 4

With the parameters shown in Table 4, the SPM is as follows:

$$SPM(k, R) = \frac{1\% \times k}{10^8} + \frac{(1\% \times 200 / 6 \times 10^{10} + 1\% / 6 \times 10^8)k}{R} + 1\% \frac{F_Size(1 - R)}{6 \times 10^{16} R}$$

Using the example above for determining the range of code rates and block size in tables 1-3, figures 8 and 9 are plotted. Figure 8 illustrates a typical image file with the size of 5MB and figure 9 illustrates an HD Video file with a typical size of 10GB.

In figure 8, line 810 illustrates SPM using code rate of 1/3 and k being 4. Line 820 illustrates SPM using code rate of 1/2 and k being 12. Line 830 illustrates SPM using code rate of 4/7 and k being 21. Line 840 illustrates SPM using code rate of 2/3 and k being 58.

In figure 9, line 910 illustrates SPM using code rate of 1/3 and k being 4. Line 920

illustrates SPM using code rate of $1/2$ and k being 12. Line 930 illustrates SPM using code rate of $4/7$ and k being 21. Line 940 illustrates SPM using code rate of $2/3$ and k being 58.

As shown in figure 8, the optimum parameters to be selected for the image should be $k=4$ and $R=1/3$. As shown in figure 9, the optimum parameters to be selected for HD Video should be $k=12$ and $R=1/2$. The results are reasonable because when the file size is large, bigger number of segments is expected to save the spatial overhead without compromising the reliability. It is also observed that the system performance in terms of the SPM can vary a lot if inappropriate parameters are selected. For example, the system performance can deteriorate 10 times if the wrong code rate e.g. $2/3$ is applied for 5MB size of file.

Figure 10 shows the reliability comparison between LDPC-based method and replication-based method for adding redundancy. As shown in figure 10, LDPC-based method can achieve higher reliability with much lower spatial redundancy. For example, to achieve the same failure rate, the spatial overhead is reduced from 200% to 43% if LDPC is applied. This means more than 50% savings of storage space can be achieved because the total space required is reduced from 300% to 143% of the original file or the system capacity can be doubled if the storage space remained the same. Further, reducing the actual physical storage directly leads to the reduction of the numbers of server required. At the same time, the corresponding power can also be conserved. On the other hand, if both methods apply 65% overhead, the failure rate is reduced from 0.05 to 10^{-13} if LDPC is applied. Furthermore, by incorporation of the invention, the system resources within the cloud such as bandwidth and processing power are consumed 10 times more balanced than the traditional method. More balanced consumption of system resources is a key to guarantee the desired maximum system capacity as it can avoid the situation that one type of resource is used up while other resources remain un-used for a large portion.

The above is a description of embodiments of a system and method in accordance with this invention. It is envisioned that others can and will design alternative methods and systems as set forth in the following claims based upon this disclosure.