Spread-Spectrum Computation

Derek Murray

University of Cambridge Computer Laboratory

The idle cycles of the world's desktop computers comprise a formidable computational resource. Attempts to harness these have hitherto been limited to *embarrassingly-parallel* algorithms, which involve the processing of independent tasks. We note that there are a wide range of data-parallel algorithms, and introduce Spread-Spectrum Computation as a technique for executing these in a distributed manner.

Data-parallel algorithms are usually targeted at supercomputers, which have hundreds or thousands of processors, and provide either a shared memory, or message passing through a fast interconnect. The distributed environment differs in three crucial respects:

- The processing nodes and network links are heterogeneous
- A processing node may fail at any time, either because it crashes, or because its owner withdraws it from the computation
- The network link to/from and between the processing nodes has much longer latency and smaller bandwidth (than a supercomputer interconnect), and it may fail at any time

Embarrassingly-parallel algorithms can cope with these problems using a simple *task-farming* approach. Heterogeneity is addressed by implicit load-balancing: each processing node requests a task when it is ready to accept one. Failure is addressed by assigning tasks to more than one processing node. These techniques only work when the tasks are independent.

Data-parallel algorithms introduce at least one dependency: all processing nodes must synchronise before the final result may be computed. There may also be intermediate exchanges of data between processing nodes (e.g. halo swaps or systolic flows). Therefore the various sub-tasks of a data-parallel job must be *coscheduled*. Spread-spectrum computation introduces techniques that enable efficient coscheduling in the distributed environment, where jobs can be affected by heterogeneity and node failures.

Why "Spread Spectrum"?

Spread-spectrum techniques are typically used in radio communications to provide security, and robustness to interference and jamming. To do this, the original signal is modulated to "hop" between pseudo-randomly selected carrier frequencies: if the sender and receiver share the same seed, they can communicate. Of course, some redundancy is required in the event that two senders hop on to the same carrier frequency at the same time.

Spread-spectrum computation involves pseudo-randomly selecting computational resources (*below*), and redundantly encoding algorithmic inputs (*right*), in order to provide robustness against delays and node failure.

Computation Dispersal Algorithms

We introduce Computation Dispersal Algorithms (CDAs) by analogy with Rabin's Information Dispersal Algorithm (IDA) [4]. The IDA encodes a file containing m blocks into $n > m$ blocks such that any m encoded blocks are sufficient to reconstruct the original file. A CDA encodes the input to a data-parallel algorithm comprising m pieces into $n > m$ pieces, such that executing the algorithm on any m encoded pieces is sufficient to reconstruct the result of applying the algorithm to the original input.

Consider the dataflow in a data-parallel algorithm. First, the coordinator decomposes the input, and distributes it to the processing nodes. The processing nodes operate on their respective pieces of input, before sending their results back to the coordinator. Finally, the coordinator reduces the results into a combined result, and returns that as output.

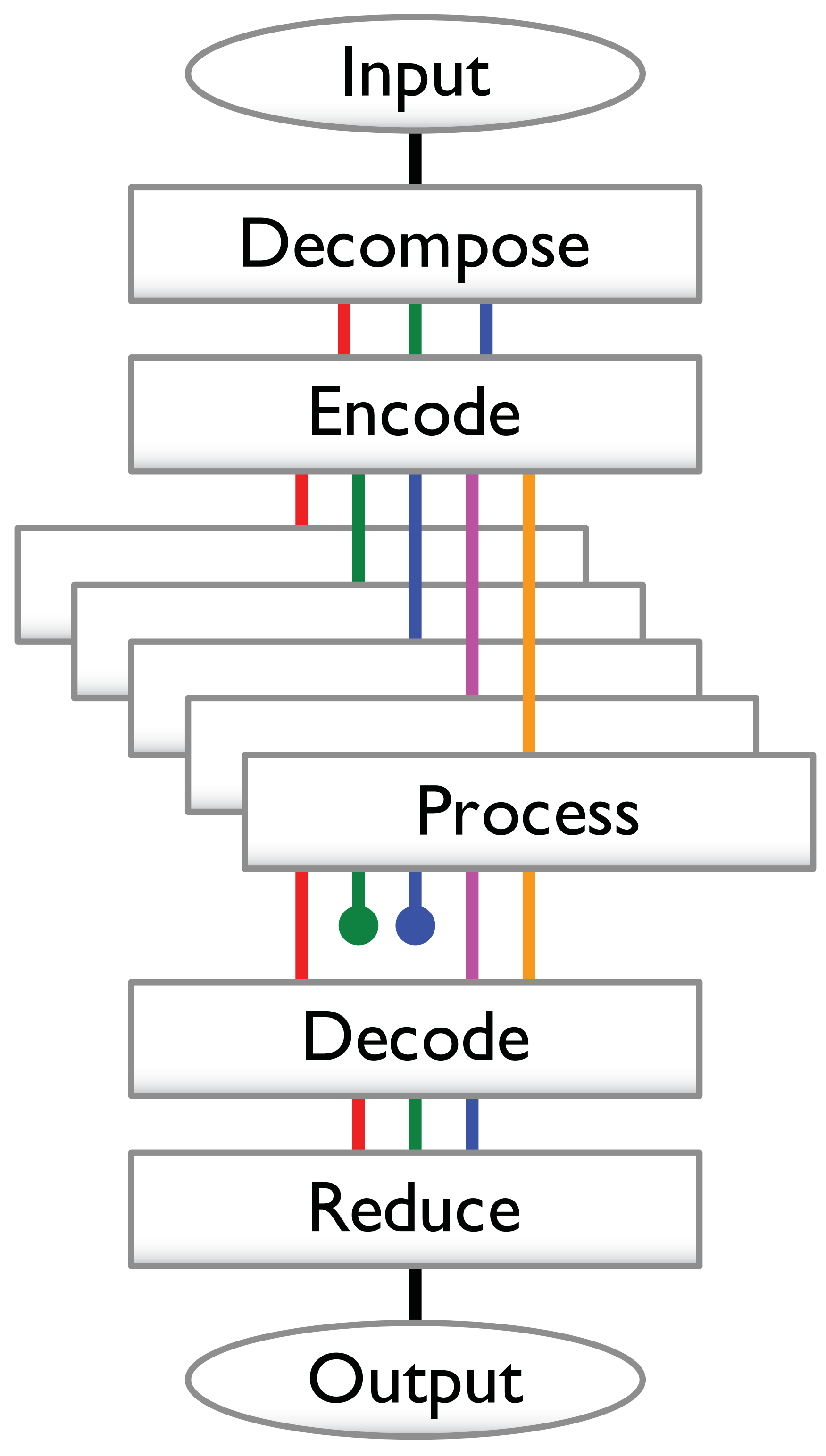
The spread-spectrum dataflow (*right*) is obtained by adding *encoding* and *decoding* steps to the above process. After the input has been decomposed into pieces, these pieces are encoded to create redundant pieces, which are also distributed to processing nodes. Since some of the nodes may fail (the green and blue paths in the diagram), the decoding step combines the successful partial results to generate the partial results that would have been obtained by running the algorithm on the original decomposition.

For many algorithms, these additional steps are sufficient to add failure and latency tolerance. Neither the kernel of the processing step nor the reduction step need be modified. Therefore it is possible to use the same highly optimised routines that form the basis of existing parallel algorithms, such as LAPACK and FFTW.

CDAs, like Rabin's IDA, are an example of erasure coding. Therefore, we can exploit the wide range of existing codes to create new CDAs. The simplest code is probably parity checking. In communication, this involves adding an additional bit to ensure that the number of 1's in a data word is either even or odd. Thus a missing bit can be calculated from the parity, as only one option can satisfy the parity condition. This notion can, for example, extend to matrices as follows:

$$A = \begin{bmatrix} \boldsymbol{a_1}^{\mathrm{T}} \\ \vdots \\ \boldsymbol{a_m}^{\mathrm{T}} \end{bmatrix} \qquad \boldsymbol{p} = \textstyle\sum_{i=1}^{m} \boldsymbol{a_i}$$

$$A' = \begin{bmatrix} A \\ \boldsymbol{p}^{\mathrm{T}} \end{bmatrix}$$



Matrix A' is the parity-enhanced version of matrix A, as it contains an additional row that is the sum of all rows in A. If A' is distributed to $m + 1$ processing nodes to compute a matrix-vector multiplication, and one node fails, the missing value in the result can be computed from the other values. This approach generalises to other linearly-separable functions, including many other matrix operations and signal processing algorithms.

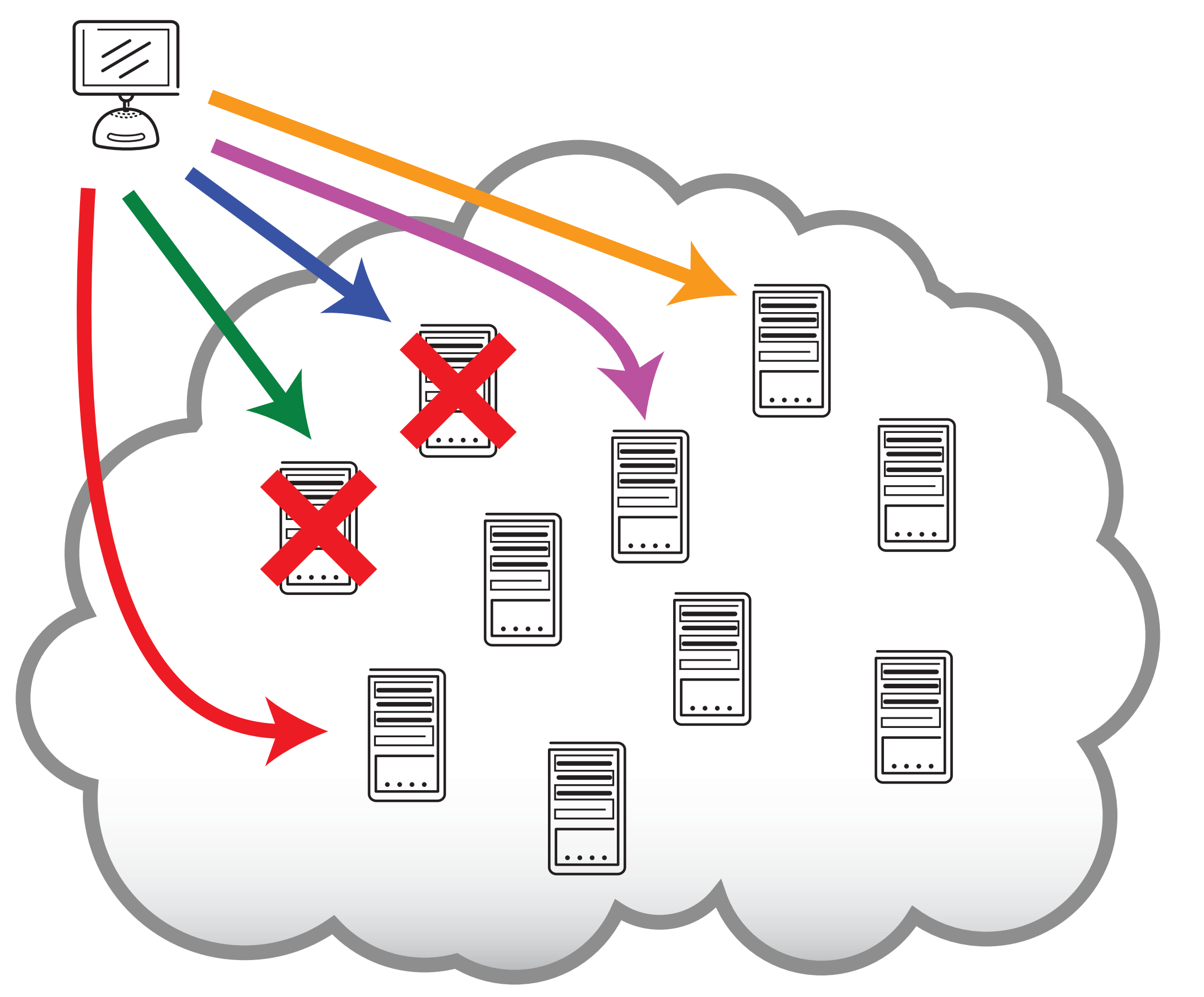
Obviously, the single parity check gives resilience to only a single failure. However, this technique works with more sophisticated codes, such as low-density parity-check (LDPC) codes [3], Tornado codes, Online codes, etc. These codes make it possible to vary the rate (i.e. m/n) of the encoding, and adapt the CDA to the observed failure distribution of the processing nodes.

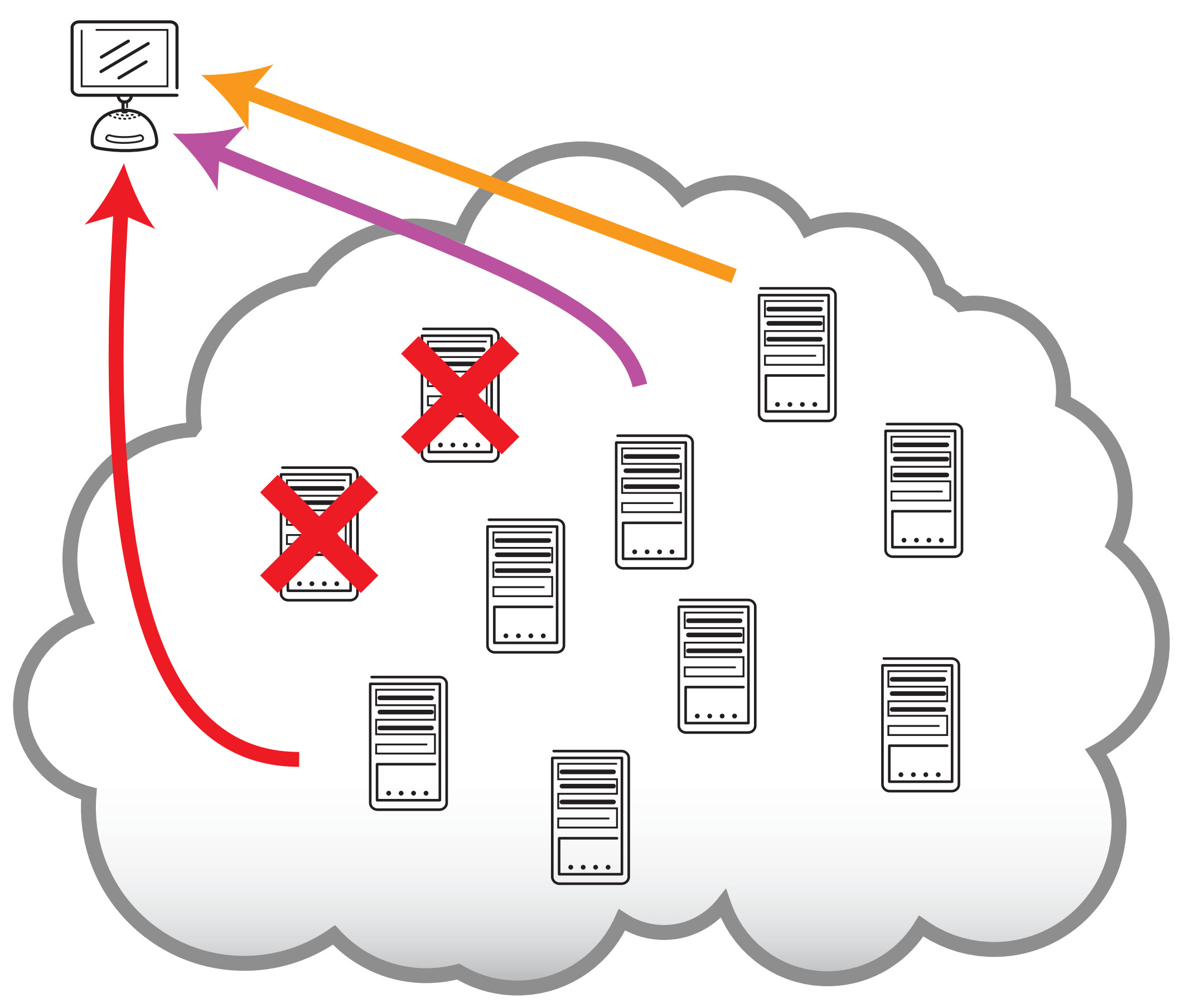
Distributed Random Scheduling

In an internet-scale distributed system, it is infeasible to track centrally the state of each processing node. More importantly, it is difficult to make a coscheduling decision when there are (i) multiple jobs under submission, and (ii) the processing nodes are heterogeneous. Therefore, we take a very simple approach: the job submitter chooses n processing nodes at random, and distributes one piece of input to each node.

Of course, this approach is vulnerable to scheduling clashes. A processing node can take one of three possible actions: run the jobs concurrently (e.g. on an idle core), queue the newly-arrived job, or reject the newly-arrived job. In the latter two cases, this appears to the submitter as additional latency or a failure, respectively. Fortunately, the use of a CDA ensures that the job will still succeed, as long as the number of failures does not exceed the number of redundant nodes.

The CDA also ameliorates the heterogeneity in the processing nodes. If the algorithm were straightforwardly distributed, the execution time would be limited by the slowest node. However, because the algorithm can complete after receiving m partial results, the performance is limited by the $(n - m + 1)^{\text{th}}$ slowest node (assuming no failures). The interplay between heterogeneity, efficiency, failure tolerance and redundancy will be the subject of further investigation.





At present, we are conducting a pilot study using PlanetLab node failure data to establish the need for spread-spectrum computation. We have obtained node liveness and performance information covering a period of one month, and we are running simulations of various job submission patterns. The key metrics are success rate, speed and system capacity.

We have started to implement spread-spectrum versions of several matrix algorithms. At present, we are concentrating on the algorithms that calculate the singular value decomposition and eigendecomposition of large, sparse matrices. These algorithms are particularly interesting, because they have applications in large-scale information retrieval [1, 2], where the use of distributed processing has become necessary.

Beyond this, we intend to look into signal processing algorithms, such as the Fast Fourier Transform, which also have applications in molecular dynamics.

Ultimately, we intend to build, deploy and measure a large-scale implementation of our spread-spectrum algorithms, and compare them with the best existing algorithms.

References

1. S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Search Engine. In *Proceedings of the seventh international conference on the World Wide Web*, pages 107-117, 1998.
2. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
3. R. G. Gallagher. *Low-Density Parity-Check Codes*. MIT Press, 1963.
4. M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989.

Email: Derek.Murray@cl.cam.ac.uk Web: http://www.cl.cam.ac.uk/~dgm36/