# Algorithm-based Fault Tolerance Applied to P2P Computing Networks

Thomas Roche
*CS, Communication&Systems*
*Le Plessis Robinson, France*
*Email: thomas.roche@imag.fr*

Mathieu Cunche
*INRIA EPI Planete*
*Grenoble University, France*
*Email: mathieu.cunche@inria.fr*

Jean-Louis Roch
*INRIA EPI Moais / LIG*
*Grenoble University, France*
*Email: jean-louis.roch@imag.fr*

***Abstract*—P2P computing platforms are subject to a wide range of attacks. In this paper, we propose a generalisation of the previous disk-less checkpointing approach for fault-tolerance in High Performance Computing systems. Our contribution is in two directions: first, instead of restricting to 2D checksums that tolerate only a small number of node failures, we propose to base disk-less checkpointing on linear codes to tolerate potentially a large number of faults. Then, we compare and analyse the use of Low Density Parity Check (LDPC) to classical Reed-Solomon (RS) codes with respect to different fault models to fit P2P systems. Our LDPC disk-less checkpointing method is well suited when only node disconnections are considered, but cannot deal with byzantine peers. Our RS disk-less checkpointing method tolerates such byzantine errors, but is restricted to exact finite field computations.**

***Keywords*-ABFT; P2P; distributed computing; SUMMA; linear coding; fault-tolerance**

## I. Introduction

Gathering thousand of resources, global platforms such as peer-to-peer (denoted P2P) provide large scale computing systems for both data storage and computational power (e.g. BOINC [1]). However, such a platform may be subject to various kinds of faults: in this paper we focus on fail-stop (e.g. peer disconnection) and by value errors [2]. In specific contexts, the possible byzantine behaviour of a peer may be statistically modelled. However, since a P2P platform usually operates in an unbounded environment, it may be the victim of orchestrated attacks against widespread vulnerabilities of specific operating systems. Therefore, in the worst case, malicious attacks have to be considered [3] that make invalid any quantified model about the attack. Fault-tolerance in such contexts has been widely studied and most of the results are dedicated to Byzantine Agreement and Leader Election algorithms [4], [5], Although our results enjoy less generality we investigated a more efficient way to tolerate byzantine peers based on coding theory ideas instead of classical voting methods. Nevertheless our results are not dedicated to specific overlay networks, the network geometry and peers hierarchy is not considered here as the our study stays at the application level. Algorithm-based fault tolerance (ABFT), introduced by K.H. Huang et al [6], was originally dedicated to matrix operations, for which it provides an efficient solution to fault tolerance in terms of time, space and complexity. It is based on the joint design of a coding technique together with a suited algorithm; due to linear algebra composition, ABFT for matrix operations is based on linear codes. Matrix operations are especially of interest in the context of large scale simulations. The use of ABFT in the context of global computing platforms has been proposed by J.L. Roch et al. [3] for exact computations. For high-performance computing (HPC) of matrix multiplication, G. Bosilca et al. [7] propose a new ABFT method for matrix multiplication based on a parity check coding of rows and columns coupled to a SUMMA [8] algorithm. It is presented in Section 2; its major interest is to consistently perform the computation with disk-less checkpointing of intermediate results. But, this technique is suited to a medium scale HPC machine: the use of parity check tolerates at most either one by-value or three fail-stops errors. Focusing on large scale and P2P platforms, the first contribution of this paper

(Section 3) is to generalize this technique to any linear code to efficiently tolerate a large number of errors. Two linear codes are then proposed: low density parity check (LDPC) codes [9] for floating point computations (Section 4) and Reed-Solomon codes that resist to malicious attacks for exact computations (Section 5). To our knowledge this paper is the first application of LDPC based ABFT for P2P platforms.

## II. Related results on Algorithms-Based Fault Tolerance (ABFT) for High Performance Computing

*A "consistent" ABFT scheme for matrix multiplication:* Let us recall the ABFT scheme proposed in [7] for matrix-matrix multiplication. It is based on two ideas: when the input matrices contain checksum rows and columns then, in absence of error, the output matrix possesses the valid checksum rows and columns; besides, at any point during the multiplication computation, the checksums are consistent (i.e. the intermediate matrices also possess valid checksum rows and columns). Hence, if $A$ and $B$ are the two input matrices and $C_R$ and $C_C$ be the checksum matrices, the respective encoded matrices $A_F$ and $B_F$ are :

$$A_F = \begin{pmatrix} A & AC_R \\ C_C^T A & C_C^T AC_R \end{pmatrix} \text{ and } B_F = \begin{pmatrix} B & BC_R \\ C_C^T B & C_C^T BC_R \end{pmatrix}$$

The equation 1 proves the consistency of checksums for the whole multiplication:

$$\begin{pmatrix} A \\ C_C^T A \end{pmatrix} \times \begin{pmatrix} B & BC_R \end{pmatrix} = \underbrace{\begin{pmatrix} AB & ABC_R \\ C_C^T AB & C_C^T ABC_R \end{pmatrix}}_{(AB)_F} \quad (1)$$

Moreover, the multiplication is performed in its outer product version (`for k=1:n,` $C_k = C_k + A_{:,k}B_{k,:}$ `end`): so the intermediate matrices $C_k$ are consistent with respect to the checksum matrices $C_R$ and $C_C$. Figure 1 details the product with consistent checksum on $3 \times 3$ matrices.

The Scalable Universal Matrix Multiplication Algorithm (SUMMA, [8]), is an efficient parallel matrix-matrix multiplication: it is widely used (e.g. in ScaLAPACK [10]).

Let us consider $p$ processes performing the matrix-matrix multiplication as a square grid of $\sqrt{p}$-by-$\sqrt{p}$ processes. Let $n \times k$ (resp. $k \times m$) be the size of $A_F$ (resp. $B_F$). Columns of $A_F$ (resp. lines of $B_F$) are treated by blocks of size $nb$, where $nb$ is a tunable parameter. The matrix $A_F$ (resp. $B_F$) is divided in blocks of size $\sqrt{p}$-by-$\frac{k}{nb}$ (resp. $\frac{k}{nb}$-by-$\sqrt{p}$). So, at the $i-th$ step of the multiplication (as described on Figure 1), $\forall j$ the $(j,i)-th$ block of $A_F$ is broadcasted to the $j-th$ row of processes and the $(i,j)-th$ block of $B_F$ is broadcasted to the $j-th$ column of processes. Then each process performs a local matrix-matrix multiplication of size $\frac{n}{\sqrt{p}}$-by-$nb$-by-$\frac{m}{\sqrt{p}}$ that it adds to its local block. The computation ends after $\frac{k}{nb}$ iterations.

After each of the $nb$ outer product iteration a global synchronisation is performed to assure data consistency. The $nb$ parameter can be tuned from 1 (one consistent state after each computed row) to $n$ (only one consistent state at the end of the computation).

Note that all the redundant computations are performed by the processes related to the last row of blocks in $A_F$ and the last column of blocks of $B_F$ since they contain only checksums data. Hence the fault tolerance cost in number of processes is $2 \times \sqrt{p} - 1$. Furthermore, if a process crashes during the $i-th$ iteration of the algorithm, its state can be recovered from the redundant information computed by those processes. Actually, due to the

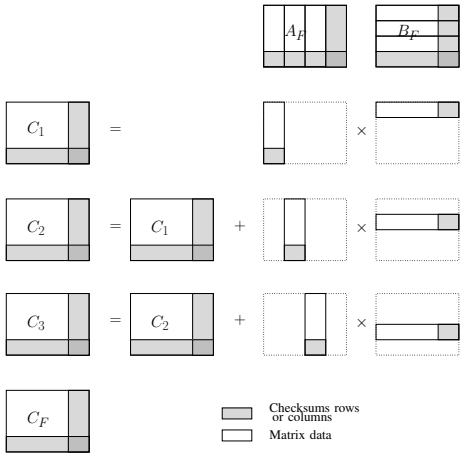


Figure 1. Checksum consistent outer product

2D checksum, up to 3 process crashes during the same iteration can be recovered.
An interesting remark is the strong scalability of this scheme with respect to the fault-tolerance overhead : the proportion of processes used for fault-tolerance ($\frac{2\times\sqrt{p}-1}{p}$) decreases when the total number of processes increases.

*Result certification:* It is important to note here that matrix-matrix multiplication enjoy an efficient post-condition check. Whereas working in finite field or floating-point arithmetic, it is always possible to verify the result $C = AB$ with high probability by randomly choosing a vector $x$ and checking if $Cx = A \times Bx$. The complexity of this check is equal to three matrix-vector product ($O(n^2)$) which is negligible compared to the cubic complexity of matrix-matrix multiplication. Furthermore, any of the intermediate $C_k$ of the SUMMA algorithm can be certified the same way since $C_k = A_k B_k$, where $A_k$ (resp. $B_k$) is the matrix $A$ (resp. $B$) only composed of the first $k$ blocks of $nb$ columns (resp. blocks of $nb$ rows).

*Limitations:* As the size of the computing platform increases, the number of processes errors and crashes increases as well. However, the described ABFT scheme is limited by the correction capability of the 2D checksums : 3 processes crashes and 1 bit-flip error. As a consequence, it is not suited to P2P networks that are subject to frequent process failures and/or disconnections.

## III. Generalisation to linear codes-based consistent ABFT, extension to P2P computing networks

To fit to P2P platforms we propose a generalisation of the previous ABFT scheme. Observing that the checksum is in fact a particular linear block code, we propose to generalize this system by using more flexible linear code in order to increase the number of errors and failures that can be tolerated by the system.

### A. Linear block codes and consistent matrix multiplication

Linear block codes are a class of codes used for error detection and correction. By adding redundancy to the source data they are able to detect and correct errors and/or erasures. A linear block code $\mathcal{C}$ of dimension $n$ and length $n+r$ is a linear subspace of dimension $n$ of the linear space $\mathbb{F}^{n+r}$, where $\mathbb{F}$ is a field. The quantity $R = n/(n+r)$ is called the code rate. The elements of $\mathbb{F}$ are called *symbols*, and the elements of $\mathcal{C}$ are called *codewords*.

The correction capabilities of a linear code are usually estimated by a characteristic called minimum distance. The minimum distance (noted $d_{min}$) of a linear block code is the minimum number of non null symbols in a codeword (excluding the zero codeword). A code with a minimal distance $d_{min}$ will be able to detect $d_{min} - 1$ and correct $\lfloor \frac{d_{min}-1}{2} \rfloor$ errors. More generally, a codeword corrupted with $t$ errors and $e$ erasures can be recovered if $d_{min} \geq 2t + e + 1$. Maximum Distance Separable (MDS) codes are such that $d_{min} = r + 1$: their distance is maximum with respect to the number $r$ of redundant symbols.

The coding scheme proposed in [7] can be easily generalised with any linear code whose symbols field $\mathbb{F}$ is compliant with the matrix entries arithmetic. The 2-dimensional parity check $C_C$ and $C_R$ are replaced by the generator matrices of chosen linear code. Thanks to the algebraic structure of a linear code, the product (as well as the sum) of encoded matrices is still a consistent encoded matrix. In all the sequel, for the sake of simplicity, we consider only the coding of the columns by $C_C$; the coding of the rows by $C_R$ is symmetric.

### B. A linear code-based consistent ABFT

Thanks to the previous generalisation, a consistent matrix multiplication can be built to tolerate a large number of errors, by tuning the redundancy of the chosen linear code.

*Scheme extension for HPC:* We consider a matrix product over a large scale HPC platform. Let's consider the product of $A$ and $B$ with SUMMA as presented in Section II and a linear code of dimension $n$ and length $n + r$, its code rate is therefore $R = n/(n+r)$. The $n \times k$ matrix $A$ is encoded into a $(n + r) \times k$ matrix $A_F$.

Like in Section II, let us consider $p$ resources viewed as a $\sqrt{p}$-by-$\sqrt{p}$ grid. The number of faults

that can be tolerated between each iteration of the SUMMA algorithm depends on the amount of redundancy added by the code. In the case of an MDS code, by adding a redundancy of $r$ symbols, the system will be able to correct $e$ erasures and $t$ faults while $r = d_{min} - 1 \geq 2t + e$. Since a process is in charge of $(n + r)/\sqrt{p}$ symbols of a column, a single node failure (fail-stop error) will cause the erasure of the same amount of symbols, and a single byzantine node (by-value error) will corrupt up to this number of symbols. In order to tolerate $E$ node failures (represented by erasures) and $T$ Byzantine nodes we need to have: $r \geq 2T(n + r)/\sqrt{p} + E(n + r)/\sqrt{p}$; so $r \geq \frac{n(2T+E)}{\sqrt{p}-(2T+E)}$.
Thus the proportion of processes used for fault-tolerance is: $\frac{(r\sqrt{p}/(n+r))\sqrt{p}}{p} = r/(n + r) = 1 - R$. It depends only on the code rate. This system is therefore weakly scalable, but as opposed to [7], has a constant code rate, which is critical to tolerate errors on P2P computing platforms.

*Adaptation to P2P computing networks:* Yet, when considering matrices of high dimensions, linear code based-ABFT is very attractive to be able to correct bit-flip errors, to recover lost data and to certify intermediate computations. All these operations are performed in $\tilde{O}(n^2)$ operations when the overall product needs $O(n^3)$. The outer product version of matrix-matrix multiplication will give us a chance to do all this by allowing an efficient fault-tolerant checkpoint scheme sketched in figure 2.

Let consider a user $U$ proposing a matrix-matrix multiplication to a P2P computing network. The property of disk-less checkpointing, which is based on a global synchronization of all participating resources, is not suited to P2P networks where all data ($A_F$, $B_F$ and the matrices $C_k$) are shared among the peers. After the data distribution through the whole network by peers downloading and uploading part of the input matrices data and tasks (which are, of course, linked to blocks of data), peers will work on the local data they possess. A fault-tolerant checkpoint is assured by sending the intermediate computations of block to the user $U$ (after each iterations or after a fixed number of iteration). When $U$ has received enough blocks corresponding to matrix $C_k$ of iteration $k$, he can decode $C_k$ thus recovering the potentially missing blocks and/or bit-flip errors. Then $U$ performs a post-condition checking on the decoded value in order to certify this state $C_k$. If, for any reason, the post-condition is not passed, he can send a stop message to the peers and resume the computation from the last certified state.
Note that the outer product integrates nicely in this P2P protocol. Indeed, due to the highly parallel algorithm, coarse grain tasks can always be divided in smaller tasks up to the finest grain (elements addition or multiplication), hence allowing a very flexible online task division and repartition (suitable to work stealing for instance [3]). Moreover, there is no need for the user to divide the input matrices in blocks with respect to the number of resources (which would be impracticable in P2P context anyways): for instance an online recursive splitting of the matrix by work-stealing will automatically load balance the computation among the peers.

## IV. LDPC-consistent ABFT on trusted P2P computing networks

We consider a P2P network where peers can leave before completing their tasks or return erroneous results. This case can be addressed by the use of Low density parity check (LDPC) codes [9] which name comes from the small number of non-null entries in the related parity check matrix. Even if their minimal distance is small, those codes show error correction capabilities close to channel capacity when decoded with a linear complexity message passing algorithm.

Since LDPC codes are in fact a concatenation of several simple parity check as used in [7], the numerical properties of the floating point checksums are still valid for LDPC codes when recovering from erasures. Yet, to deal with fault values, LDPC codes are restricted to exact arithmetic.

Using an LDPC-based ABFT enables to tolerate a number of peers disconnections and non-malicious by-value errors close to the optimal with a high probability (see Table.I). The algorithm used for error correction runs in $O(n)$ [9], so in the same

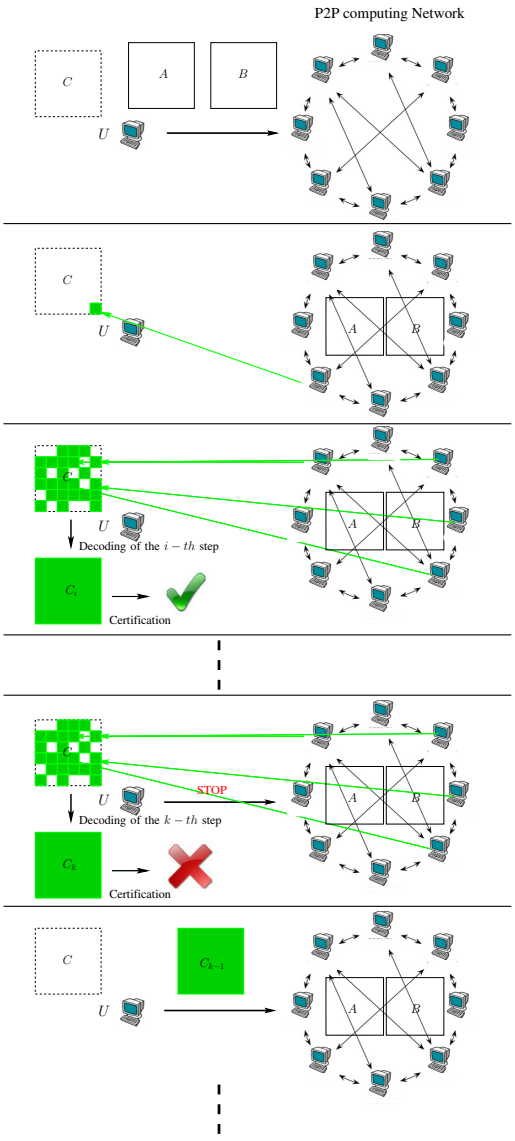


Figure 2. Fault tolerant checkpoint protocol

order of complexity as the 2D checksums scheme of [7]: decoding the whole matrix requires the decoding of $n$ codewords with a $O(n^2)$ complexity.

## V. RS-consistent ABFT on untrusted P2P computing networks

We now consider the byzantine fault model in the context of P2P networks that are subject to malicious attacks. Indeed in a P2P Network, the peers cannot be trusted. This limits the confidence in the outputted result. While the matrix-matrix multiplication possesses an efficient post-condition (see Section II), one has yet to be sure the post-condition checking was not forged by a malicious peer. Furthermore, assuming the post-condition checking can be trusted, using the fault-tolerance studied in the previous section would allow very cheap Denial of Service attacks. The small minimal distance of LDPC codes means that one can forge a valid codeword with only few modified coordinates (i.e. few peers controlled by an attacker). Codes with large minimum distance will better fit to this case (e.g. MDS codes). Let us assume a part of the available computing platform can be trusted. The size (so the computational power) of this trusted platform is expected to be very small compared to the untrusted platform. The trusted part should be used for decoding and, more importantly, post-condition checking (e.g in figure 2 user $U$ is trusted and the P2P computing network is not).

*Reed-Solomon:* Reed-Solomon (RS) are MDS codes introduced in 1960 by I. Reed and G. Solomon. RS codes are always able to correct $r/2$ errors (when it's only under a certain probability of error for LDPC codes) while having a decoding complexity barely higher than LDPC ($O(nlog^2n)$ [11], see Table I). RS codes are well suited to tolerate Byzantine (so malicious) errors with exact arithmetic. Thus, with large enough redundancy, the use of RS-based ABFT for consistent matrix multiplication with exact arithmetic, prevents non massive malicious attacks [3]: forging a wrong result would require the attacker to control many peers. However, the non compatibility of underlying arithmetic makes RS-based ABFT not suited to floating point computations.

## VI. Conclusion and future work

In this paper, we presented a generalization of consistent ABFT matrix multiplication based on linear codes which is suited to P2P computing. The proposed protocol enables to tolerate the potential erratic behaviour of the P2P network. Considering exact arithmetic, the use of RS code provides

| | RS | LDPC |
|---|---|---|
| Overhead | $r/n$ | $r/n$ |
| Recovery complexity (per column) | $O(nlog^2(n))$ | $O(n)$ |
| Recovery complexity (whole matrix) | $O(n^2log^2(n))$ | $O(n^2)$ |
| Fault tolerance (errors per column) | $\lfloor r/2 \rfloor$ | $15\%$ of $n+r$ confidence $1-10^{-6}$ $n = r = 0.5 \times 10^6$ [12] |
| Fault tolerance (erasures per column) | $\lfloor r/2 \rfloor$ | $49\%$ of $n+r$ confidence $1-10^{-4}$ $n = r = 0.25 \times 10^7$ [13] |

Table I
COMPARISON OF RS AND LDPC CODES FOR ABFT

efficient fault tolerance not only in case of peer disconnection but also in case of byzantine errors (even considering malicious peers). For floating point arithmetic computations, the use of LDPC codes supports an arbitrary number of fail-stop errors with efficient coding and decoding.
Our perspective is to use this protocol for an effective challenge in exact arithmetic computation.

## REFERENCES


[1] D. P. Anderson, "Boinc: a system for public-resource computing and storage," 2004, pp. 4–10. [Online]. Available: http://dx.doi.org/10.1109/GRID.2004.14

[2] A. Avizienis, J.-C. Laprie, and B. Randell, "Dependability and its threats - a taxonomy," in *IFIP Congress Topical Sessions*, R. Jacquart, Ed. Kluwer, 2004, pp. 91–120.

[3] J.-L. Roch and S. Varrette, "Probabilistic certification of divide & conquer algorithms on global computing platforms. Application to fault-tolerant exact matrix-vector product," in *Parallel Symbolic Computation'07 (PASCO'07)*, A. publishing, Ed., London, Ontario, Canada, July 2007.

[4] V. King, J. Saia, V. Sanwalani, and E. Vee, "Towards secure and scalable computation in peer-to-peer networks," in *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 87–98.

[5] B. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani, "Fast asynchronous byzantine agreement and leader election with full information," in *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008, pp. 1038–1047.

[6] K.-H. Huang and J. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. 33, no. 6, pp. 518–528, 1984.

[7] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, "Algorithmic based fault tolerance applied to high performance computing," *CoRR*, vol. abs/0806.3121, 2008.

[8] R. A. van de Geijn and J. Watts, "Summa: scalable universal matrix multiplication algorithm," *Concurrency - Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.

[9] R. G. Gallager, "Low density parity check codes," Ph.D. dissertation, MIT, Cambridge, Mass., September 1960.

[10] J. Dongarra, K. Madsen, and J. Wasniewski, Eds., *Applied Parallel Computing, Computations in Physics, Chemistry and Engineering Science, Second International Workshop, PARA '95, Lyngby, Denmark, August 21-24, 1995, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1041. Springer, 1996.

[11] R. E. Blahut, *Theory and Practice of Error Control Codes*. Addison-Wesley.

[12] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[13] H. D. Pfister, I. Sason, and R. Urbanke, "Capacity-Achieving Ensembles for the Binary Erasure Channel with Bounded Complexity," in *Fourth ETH–Technion Meeting on Information Theory and Communications*, 2004.