إعادة بناء مخطط تدفق البيانات لضمان سلامة تشغيل التطبيقات المتوازية الموزعة

د. سمير جعفر
قسم الرياضيات – كلية العلوم – جامعة دمشق – سوريا

الملخص

يقع هذا البحث ضمن مجال التسامح مع الأعطال في البيئات التفرعية الواسعة مثل الحوسبة الشبكية حيث تتميز هذه البيئات بالعدد الكبير لمكوناتها الموزعة جغرافياً، حيث تقوم على مبدأ تشارك عدد كبير من العقد الحسابية لتنفيذ تطبيق متوازي ولكن كلّ ذلك الاتساع والعدد الهائل للعقد المشاركة في النظام قد أنشأ عقبات كبيرة لتحقيق سلامة التشغيل في هذه البيئات الواسعة. لذلك تم التوجه في هذا البحث إلى إيجاد آلية جديدة للتسامح مع الأعطال للتطبيقات المتوازية تقوم على مبدأ إعادة بناء حالة تنفيذ التطبيق بعد العطل بهدف الحفاظ على استمرارية عمله وكذلك لضمان انتهاء تنفيذ تلك التطبيقات في ظل وجود الأخطاء الناتجة عن عطل أو مغادرة العقد أو انقطاعها عن شبكة الاتصال في لحظة ما خلال التنفيذ، وقد استخدمنا مخطط تدفق البيانات كنموذج مجرد لتمثيل حالة تنفيذ التطبيق.

الكلمات المفتاحية: الحوسبة الشبكية، نقطة الاستعادة، الاسترجاع، البرمجة المتوازية، مخطط تدفق البيانات، سرقة العمل، سلامة التشغيل، التسامح مع الأعطال.

Rebuilding Dataflow Graph for Dependability on Parallel distributed Applications

Dr. Samir Jafar

Department of Mathematics – Faculty of Science – Damascus University – Syria

Abstract

The study is researching the fault tolerance in the large distributed environments such as Grid-Computing, where characterized by the large number of their components and geographic breadth. However all that widening and the huge number of the nodes involved in the system has created great hurdles to achieve dependability in these large environments. So the orientation of this research is to a new mechanism of fault tolerance in the large scale and distributed environments where are based on the principle of rebuilding dataflow graph of the application in order to ensure the continuity of the application as well as to ensure the completion of execution in presence of faults, which resulting from the leave of the nodes and the interruption of the network at a given moment during the execution time.

Keywords: grid computing, recovery, checkpointing, parallel programming, macro data flow, work stealing, dependability, fault tolerance.

1. مقدمة:

إن التقدم الملحوظ في صناعة الحواسيب وتصميم شبكات الاتصالات خلال السنوات الماضية، سمح بتطوير البنى والنظم الموزعة والمتوازية التي تعتمد بشكل أساسي في استخدامها على مجموعة من الحواسيب المربوطة فيما بينها بشبكة أو عدة شبكات اتصال. كذلك شهدنا خلال الفترة الماضية على تطور شبكات الاتصال من شبكات محلية (LAN) قصيرة المدى إلى شبكات كبيرة (WAN) بعيدة المدى, إن التطور في هذه التجهيزات قد سمح بظهور نظم وبنى متوازية جديدة كبيرة الحجم مثل الحوسبة الشبكية (Grid–Computing) وعناقيد الحواسيب (Clusters) والأنظمة P2P. إن هذه البنى الجديدة والمختلفة عن النظم التقليدية والبنى المتماثلة ذات الذاكرة المشتركة SMP (Symmetric Multi–Processors), قد صممت خصيصاً من أجل تقديم حل جيد ومناسب لحاجات ومتطلبات الباحثين من مختلف الفروع العلمية وهذه الحاجات تتميز و تتلخص ببرامج تتطلب الكثير والكثير من القدرة على إجراء الحسابات وذلك فيما يخص زمن المعالجة وسعة الذاكرة [3] [2] [1].

إن التطور في الأدوات المتوفرة والحاجات المطلوبة من قبل المستخدمين للنظم الموزعة و المتوازية يمكن تلخيصها بما يلي :

- **التطبيقات الموزعة والمتوازية:** إن الحاجة إلى استطاعة كبيرة في الحساب أدت إلى زيادة مهمة جداً في العمل على تطوير وبرمجة التطبيقات المتوازية والموزعة. إن استخدام تقنية التوازي يسمح بتمثيل بعض الظواهر التي لا يمكن التعبير عنها بشكل حقيقي [4]، أو الحصول على حلول مثلى لمشاكل رياضية ذات حجوم كبيرة [5].
- **عدم التجانس:** إن الأجهزة والأدوات المستخدمة من قبل التطبيقات المتوازية والموزعة وذلك إن كانت أجهزة حواسيب أو بنى تحتية للاتصالات والشبكات، هي أجهزة وأدوات مختلفة وغير متجانسة. أجهزة الحواسيب يمكن أن تكون إما أجهزة متماثلة SMP أو أجهزة حواسيب شخصية أو أجهزة محمولة. كذلك فإن شبكات

الاتصال المستخدمة من أجل ربط هذه الحواسيب يمكن أن تكون إما شبكات صغيرة مثل Ethernet, Myrinet, Infini Band أو شبكات كبيرة مثل الإنترنت.

- **الديناميكية** : إن عدد الأجهزة المكونة للنظم والبنى المتوازية والموزعة ليس عدداً ثابتاً. في الحقيقة إن عدد الأجهزة المتوفرة والمشاركة في النظام يمكن أن يتغير في أي لحظة حتى خلال مرحلة تنفيذ التطبيقات.

إن الأخذ بعين الاعتبار لهذه التطورات المذكورة سابقاً يفرض شروطاً جديدة على مستوى نظم التشغيل و/أو الأدوات التي تسمح بتطوير وتشغيل التطبيقات الموزعة والمتوازية. هذه الشروط يمكن عرضها على الشكل التالي :

- **أخذ الأعطال بعين الاعتبار:** بما أن نتائج التنفيذ الموزع لأي تطبيق متوازي موزع يتعلق بأكثر من جهاز (حاسب) مربوطة فيما بينها بشبكة اتصال فإن احتمال ظهور عطل خلال تنفيذ التطبيق (البرنامج) على نظام متوازي موزع يصبح كبيراً جداً بحيث أن العالم Leslie B. *Lamport* [3] يعطي التعريف التالي لأي نظام موزع وهو يخاطب أحد زملائه: " النظام الموزع هو نظام بحيث أنه إذا حدث عطل في أحد الحواسيب المكونة للنظام فإنه لا يمكنك حتى أن تعرف متى يصبح جهازك غير مستخدم." ضمن هذه المعطيات فإن التأكد من صحة تشغيل وعمل التطبيقات هو أمر بالغ الأهمية وبالتالي فإن إدخال هذه الآلية في تصميم النظم المتوازية والموزعة أصبح ضرورة لا غنى عنها من أجل ضمان الحصول على نتائج صحيحة من.
- **أخذ عدم التجانس بعين الاعتبار:** إن الاختلاف بين التجهيزات المكونة للبنى المتوازية والموزعة قد أدخل عدم تجانس كبير داخل هذه البنى الجديدة وهذا الأمر قد حفز العلماء على استخدام تقنيات التجريد لأجهزة الحواسيب والتي تسمح بإعطاء حالة تنفيذ غير متعلقة بالجهاز المستخدم. هذه التقنيات يمكن الحصول عليها من خلال استخدام الآلات الافتراضية مثل JVM أو من خلال الوصف الرياضي لحالة تنفيذ البرامج والتطبيقات.

- **أخذ الديناميكية بعين الاعتبار:** إن عدد الأجهزة المتوفرة داخل نظام متوازي موزع يمكن أن يتغير من لحظة لأخرى وذلك يأتي من قدرة هذه النظم على استقبال أجهزة جديدة أو حذف جهاز من الأجهزة المستخدمة في حالات الضرورة مثل العطل، الصيانة أو الاستخدام الأكثر أولوية. إذن التطبيقات والبرامج المتوازية والموزعة يجب أن تكون قادرة على التكيف والتأقلم خلال كامل فترة تنفيذها مع عدد الأجهزة المتوفرة بشكل حقيقي داخل النظام.

إن إضافة حلول للمشاكل المطروحة سابقاً داخل بيئات البرمجة ونظم التشغيل المتوازية والموزعة قد أدى إلى مضاعفة الأبحاث العلمية من أجل الحصول على نظام تشغيل سهل ومضمون وجيد لبنى الحساب المتوازية والموزعة [6] [7]. هذه الأبحاث العلمية توجهت بشكل أساسي حول ما يسمى بسلامة تشغيل النظم (Dependability) من خلال التسامح مع الأخطاء أو الأعطال (Fault Tolerance) [8].

في السنوات الأخيرة قدمت عدة حلول من أجل معالجة أعطال الأجهزة والبرمجيات إلا أن الاستخدام الجيد والمضمون للبنى المتوازية والموزعة ذات الحجوم الكبيرة مازال محدوداً جداً بسبب المشاكل الناتجة عن عدم تجانس وديناميكية هذه البنى. وبالتالي فإن الأخذ بعين الاعتبار لهذين العاملين هو أمر أساسي إذا أردنا الاستفادة من قدرات واستطاعة هذه النظم الجديدة، ولهذه الأسباب تم الاهتمام بهذه المشكلة العلمية.

2. هدف البحث:

إن الهدف من هذا العمل هو تصميم آلية لضمان صحة تشغيل واستخدام البنى المتوازية والموزعة والمؤلفة من عدد متغير من الحواسيب غير المتجانسة بحيث تسمح لأي تطبيق بأن ينفذ بشكل صحيح مع ضمانة الجودة، على بنية متوازية موزعة وديناميكية، هذه الآلية يجب أن تسمح بما يلي :

1. إضافة وحذف العقد (الحواسيب) بشكل ديناميكي خلال فترة تنفيذ التطبيقات.
2. معالجة الأعطال التي تصيب بعض الحواسيب المستخدمة في النظام دون

الحاجة لإعادة تشغيل جميع حواسيب النظام أو لإعادة تنفيذ التطبيقات من البداية.

3. إمكانية متابعة عمل الحاسوب أو الحواسيب المعطلة على أي نوع آخر من الحواسيب المتوفرة حتى لو كانت غير متجانسة.
4. التقليل، داخل النظام، من عدد الحواسيب المفروضة غير قابلة للتعطل خلال كامل فترة التشغيل.
5. أن تكون الكلفة الزمنية لهذه الآلية قليلة عند عدم حدوث أعطال وأن تبقى معقولة عند حدوث الأعطال خلال التنفيذ.

3. مواد وطرق البحث:

1.3 المفاهيم الأساسية:

سنقدم في هذه الفقرة بعض التعاريف والمصطلحات والمفاهيم التي تعتبر الركيزة الأساسية للعمل في هذا البحث.

تعريف 1: المهمة (Task) هي سلسلة من التعليمات التي تنفذ محلياً وبشكل تسلسلي على معالج واحد.

تعريف 2: التطبيق المتوازي (Parallel Application) هو مجموعة من المهام ، المستقلة و/أو المترابطة، والتي يمكن أن تنفذ بالوقت نفسه على معالجات متباعدة جغرافياً بهدف تقليل زمن تنفيذ التطبيق.

تعريف 3: الحالة العامة (Global Sate) لتطبيق متوازي في لحظة ما t هي تشكيلة من الحالات المحلية لجميع المعالجات المشاركة في تنفيذ التطبيق في اللحظة t بالإضافة لحالات قنوات الإتصال بين هذه المعالجات في تلك اللحظة t [10] [9] [8].

تعريف 4: الحالة العامة المتناسقة (Consistent global state) لتطبيق متوازي هي ببساطة إحدى الحالات العامة للتطبيق المتوازي والتي تحصل خلال التنفيذ الصحيح للتطبيق (بدون أعطال). بمعنى آخر، تكون الحالة العامة للتطبيق المتوازي متناسقة إذا كانت حالة أي معالج من المعالجات المشاركة تحوي حدث استقبال لرسالة ما m فإن حالة المعالج الذي أرسل m تحوي على حدث إرسالها [10] [9] [8].

تعريف 5: الإستعادة (Recovery) يمكن للتطبيق إجراء استعادة صحيحة للتنفيذ إذا كانت حالته الداخلية متوافقة مع سلوك ملاحظ للتنفيذ قبل حدوث العطل[10] [9] [8] .

2.3 نموذج البرمجة والتنفيذ للتطبيقات المتوازية [12] [11]:

أن نموذج البرمجة المتوازية المراد دراسته في هذا العمل هو نموذج عالي المستوى حيث أن تحديد العمل المتوازي في التطبيق يتم بشكل ديناميكي خلال التنفيذ من خلال تحليل الإعتمادية لمعطيات البرنامج. هذا التحليل ينجز على مستوى المهام، وبالتالي فإن تنفيذ البرنامج يقسم ضمنياً إلى مهام حسابية من قبل المبرمج ولتحديد التوازي بين هذه المهام يلزم تحليل الاعتمادية للمعطيات بين المهام. بفضل هذا التحليل يستطيع النظام خلال التنفيذ الحصول والمحافظة على تمثيل مجرد لحالة تنفيذ التطبيق على شكل مخطط يصف المهام واعتمادية المعطيات ندعو هذا المخطط بمخطط تدفق البيانات. نظام التنفيذ يرتكز على آلة افتراضية من أجل تنفيذ مهام التطبيق المتوازي مع احترام شروط الأسبقية المفروضة من اعتمادية المعطيات. وبالتالي فإن توصيف التوازي في التطبيق مستقل عن البنية الموزعة المنفذ عليها وخوارزمية الجدولة تقرر من هي المعالجات المسؤولة عن تنفيذ المهام وتخزين المعطيات.

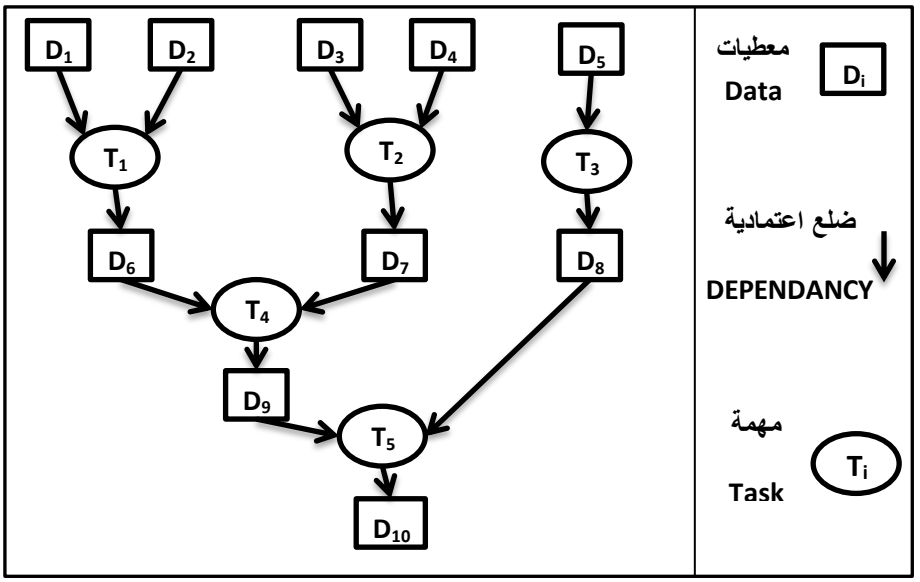
1.2.3 مخطط تدفق البيانات (DataFlow Graph):

مخطط تدفق البيانات الممثل لتنفيذ تطبيق متوازي هو بيان $G = (\nu, \varepsilon)$ حيث أن المهام ν_t والمعطيات ν_d تشكل مجموعة عقد البيان وأن وصول المهام إلى المعطيات يشكل مجموعة أضلاع البيان ε. أي عقدة "مهمة" task تكون مرتبطة بعقدة معطيات

data أو أكثر وكذلك أي عقدة معطيات data تكون مرتبطة مع عقدة "مهمة" task أو أكثر. الضلع في هذا المخطط يعني تزامن من نوع قراءة أو كتابة بين معطاة ومهمة. لتكن $t \in \nu_t$ مهمة و $d \in \nu_d$ معطاة فإن:

- الضلع $(t,d) \in \varepsilon$ يعني حق وصول بالكتابة للمهمة t على المعطاة d ؛ المهمة t تسبق أي مهمة أخرى t' لها حق الوصول بالقراءة للمعطاة d.
- الضلع $(d,t) \in \varepsilon$ يعني حق وصول بالقراءة للمهمة t على المعطاة d ؛ المهمة t مسبوقة أو تلي جميع المهام التي لها حق الوصول بالكتابة على المعطاة d.

الشكل (1) يوضح مثال لمخطط تدفق البيانات الممثل لتنفيذ تطبيق متوازي بالإعتماد على نموذج البرمجة المتوازية ونموذج التنفيذ المعتمد.



الشكل (1) : مخطط تدفق البيانات

2.2.3 سرقة العمل (Work Stealing) [13]:

بما أن التمثيل المجرد (مخطط تدفق البيانات) لحالة تنفيذ التطبيقات المتوازية المدروسة ينشأ بشكل ديناميكي خلال التنفيذ، فإن خوارزمية الجدولة المستخدمة لتوزيع المهام على المعالجات يجب أن تكون بالضرورة ديناميكية. الجدولة بواسطة سرقة العمل هي الطريقة المعتمدة في هذا البحث وتقوم على المبدأ التالي: كل معالج يضع المهام التي ينشئها في قائمة محلية، عندما ينتهي المعالج من تنفيذ مهمة ما، نميز حالتين:

- إما قائمته المحلية تحوي على مهام جاهزة للتنفيذ (واحدة على الأقل) وبالتالي يأخذ الأحدث منها وينفذها محلياً.
- أو قائمته المحلية لا تحوي على مهام جاهزة للتنفيذ وبالتالي ينتقل هذا المعالج إلى حالة السرقة ويبحث عن مهام جاهزة على المعالجات الأخرى ويبقى على هذه الحالة حتى يجد معالج ضحية لديه مهام جاهزة فيأخذ أقدمها.

3.2.3 الآلية المقترحة:

إن الآلية المقترحة لضمان سلامة تشغيل التطبيقات المتوازية في البيئة المدروسة تعتمد على إعادة بناء مخطط تدفق البيانات (الذي يمثل حالة تنفيذ التطبيق تمثيلاً مجرداً عن بيئة التنفيذ) إلى ما قبل لحظة العطل المكتشف. لإعادة بناء مخطط تدفق البيانات للمعالج المعطل يلزم تخزين البيانات الضرورية والكافية، ولضمان حالة عامة متناسقة للتطبيق يجب استعادة بعض العمليات وضمان حتمية التنفيذ، بالتالي فإن الآلية المقترحة تتكون من مرحلتين: التخزين والاستعادة. قبل تقديم هاتين المرحلتين سنقوم بشرح محتويات نقطة الاستعادة لمعالج ومكان تخزينها:

- **نقطة الاستعادة (Checkpoint):**

لضمان امكانية استبدال المعالج المعطل بمعالج آخر متوفر بغض النظر عن نوعه وبرامجه سنتجنب تخزين أي معطيات خاصة بحالة تنفيذ المعالج وسنقتصر على تخزين

البيانات المتعلقة بمهام التطبيق فقط. عندما يقوم المعالج بأخذ نقطة إستعادة سيقوم فعلياً بتخزين قائمة المهام الموجودة لديه فقط على ذاكرة مستقرة أي يمكن الوصول إليها بعد حدوث العطل، وللتعامل مع صفة قابلية التوسع (large scale) في الحوسبة الشبكية سنجعل كل عقدة تتصل بمعالج خفيف (process) خاص لنقاط الاستعادة يُستخدم كذاكرة مستقرة لمعلومات تلك العقدة ولكن يمكن لهذا المعالج أن يكون ذاكرة مستقرة لأكثر من عقدة، نسمي هذا المعالج بمخدّم نقاط الاستعادة (Checkpoint Server) بحيث يكون إمّا مركزي Centralized (مشترك لجميع العقد) أو موزّع Distributed (أي موجود كذاكرة مستقرة على أجهزة موثوقة).

- **مرحلة التخزين:**

إن الآلية المقترحة تعتمد من جهة أولى على نقاط استعادة دورية لحالة التطبيق ضمن المعالجات ومن جهة ثانية تعتمد على بنية معطيات خاصة بكل معالج نسميها رتل العمل(work queue) تُحفظ فيها المهام المسروقة منه من جهة أخرى، حيث يُخزَّن هذا الرتل ضمن الذاكرة المستقرة. إذاً عند حدوث سرقة بين معالجين نقوم بما يلي: يُسجّل المعالج الضحية المهمة المسروقة منه ضمن رتل عمله الخاص في الذاكرة المستقرة بالشكل التالي (حيث تعبر الوسطاء عن مدخلات المهمة):

(id of task, id of thief, parameter1,…,parameter n, results)

تجدر الإشارة هنا إلى عدم توقف عملية أخذ نقطة الاستعادة خلال عملية السرقة وبالتالي عندما يحين وقت أخذ نقطة الاستعادة لأحد المعالجات فقد يُحفظ جزء من العمل(المهام المسروقة) مرتين، سنستخدم لذلك خوارزمية التخلص من المعلومات الفائضة خلال مرحلة الاسترجاع بعد حدوث عطل ما لذلك المعالج لتحسين الكلفة التخزينية خلال التنفيذ.

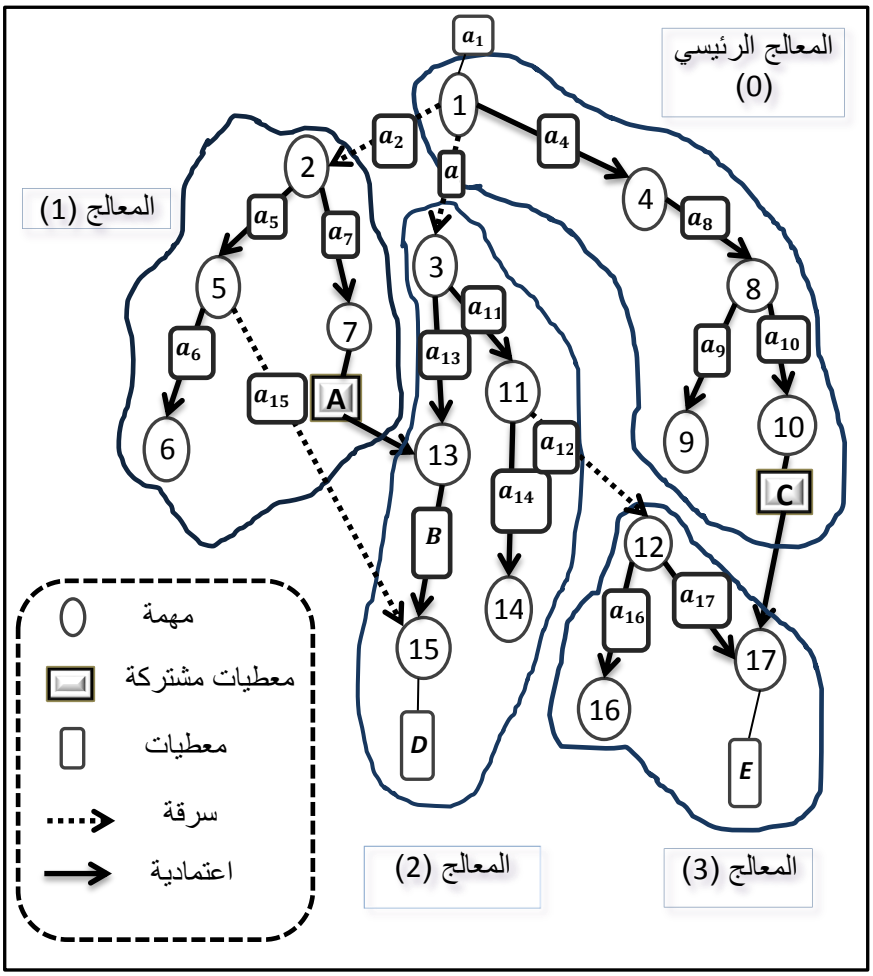
- **مرحلة الاستعادة:**

عند تعطل معالج ما أو انقطاعه عن الشبكة يتم تعميم هذا الحدث على جميع المعالجات المشاركة في تنفيذ التطبيق، وهنا نميز حالتين:

1. **المعالج المعطّل هو معالج ضحية (victim):** يحل محل المعالج المعطّل معالج بديل يحمل نفس المعرّف (id) الخاص بالمعالج المعطّل ومن ثم يقوم هذا البديل بالاسترجاع من آخر نقطة استعادة مخزّنة بالإضافة إلى قراءة رتل العمل المخزن في الذاكرة المستقرة، فإذا وجد مهمة مسجّلة بأنها مسروقة (سواءً في نقطة الاستعادة أو في رتل العمل) ولم يحصل على نتيجتها بعد فإنّه يُبقي عليها في رتل عمله بانتظار النتيجة من السارق أما المهام المنتهية والمحفوظة فسيستخدمها لإعادة بناء مخططه الجزئي كما كان قبل العطل ومن ثم يتابع التنفيذ، وبالتالي نلاحظ أن الاسترجاع يكون محلّي للمعالج المعطّل فقط.
2. **المعالج المعطّل هو معالج سارق (thief):** يحل مكان المعالج المعطّل معالج بديل حيث يقوم بالاسترجاع من آخر نقطة استعادة مخزّنة ويتابع العمل، أما بالنسبة للمهام التي كان قد سرقها المعالج المعطّل قبل العطل فإنه عند تعميم الفشل على المعالجات سيقوم كل معالج بالبحث ضمن مهامه عن مهمة قد سُرقت من قبل هذا المعالج المعطّل ولم تصل نتيجتها بعد فيقوم المعالج الضحية عندها بإعادة إرسالها للمعالج البديل ليكمل تنفيذها (على أنها مهمة مسروقة)، فإن لم يجد يتابع تنفيذ عمله إما بمتابعة تنفيذ مهامه المحلّية أو بسرقة مهام جاهزة من معالجات أخرى.

4.2.3 مثال يوضح عمل الآلية المقترحة:

سنحاول إيضاح ما ذكرناه سابقاً من خلال هذا النموذج لتطبيق متوازي مُنفّذ ضمن بيئة الحوسبة الشبكية وممثل باستخدام مخطط تدفق البيانات حيث نوضح فيه كل من مرحلتي التخزين والاسترجاع للآلية المقترحة.



الشكل (2) : مخطط تدفق البيانات لتطبيق ما خلال التنفيذ الطبيعي

يوضح الشكل (2) حالة تنفيذ تطبيق متوازي على أربعة معالجات نوضح من خلاله عمليات السرقة التي تحدث بين المعالجات في لحظة ما أثناء التنفيذ، إذاً وبحسب الآلية المقترحة سيبدأ كل من المعالجات التي في موقع الضحية بتعبئة أرتال العمل الخاصة بها كما يلي:

Id of task	Id of thief	Parameter1	Results
2	1	a_2	a_5
3	2	a_3	a_{13}

الجدول (1): رتل عمل المعالج الرئيسي

Id of task	Id of thief	Parameter1	Results
12	3	a_{12}	a_{17}

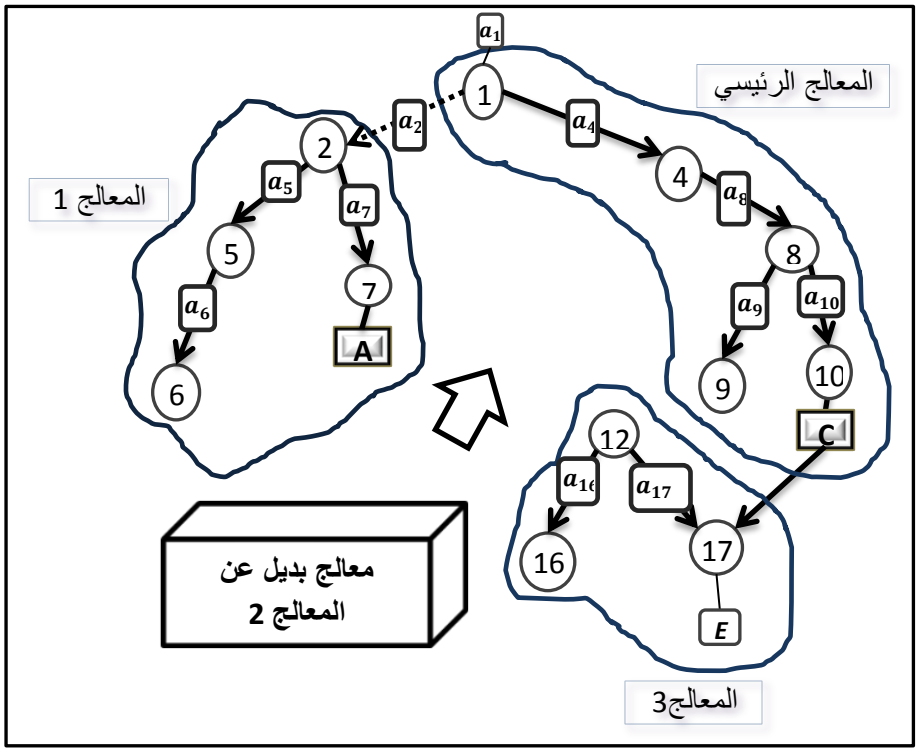
الجدول (2): رتل عمل المعالج 2

Id of task	Id of thief	Parameter1	Parameter2	Results
15	2	a_{15}	B	D

الجدول (3): رتل عمل المعالج 1

وبالتزامن مع تكوين تلك البنى وحفظها ضمن الذاكرة المستقرة فإنّ كل معالج يحتفظ بنقطة استعادة دورية خاصة به ضمن الذاكرة المستقرة أيضاً. ولمناقشة مرحلة الاسترجاع نعرض حالة تعطّل المعالج 2 أثناء التنفيذ كما يلي: بفرض تعطّل المعالج 2 خلال التنفيذ لسبب ما عندها ووفقاً للآلية المقترحة يُعزل المعالج المعطّل ويحل محلّه معالج بديل يحمل نفس المعرّف للمعالج المعطّل ليتابع عمله كما يبين لنا الشكل 3. عندها بفرض أنّ المعالج 2 كانت آخر نقطة استعادة له تشمل على تنفيذ كل من المهام (3،11،13) عندئذ يقوم المعالج البديل بالاسترجاع من آخر نقطة استعادة مخزّنة والتي تحوي على تنفيذ المهام السابقة بالإضافة إلى معلومات عن أضلاع الاعتمادية بين المعالجات الأخرى، نلاحظ أنّ المعالج 2 يقع في موقع الضحية والسارق أي تُطبّق عليه الحالتين المذكورتين أعلاه فهو ضحية للمعالج 3 حيث سُرق منه المهمة 12 المسجّلة في رتل عمله وقد حصل على نتيجتها قبل العطل إذاً فلا

يُجبِر المعالج 3 على إعادة تنفيذها وإنّما يقوم بإعادة الربط بين مخططي تدفق البيانات للمعالجين بضلع سرقة.



الشكل (3) : تعطل المعالج 2 واستبداله بمعالج آخر بديل

من جهة أخرى فهو سارق من المعالجين 1 و0 عندئذ وبحسب الخطوات الموضّحة لمرحلة الاسترجاع يجري ما يلي:

- بعد تعميم العطل على جميع المعالجات المتوفرة يقوم كل معالج بالبحث في رتل عمله الخاص عن مهام قد سُرقت من قبل المعالج 2.
- عندها يجد المعالج الرئيسي في رتل عمله المهمة 3 وقد سُرقت وسُجِّلت نتيجتها وكذلك الحال بالنسبة للمعالج 1 ومهمته المسروقة 15 .

- وبالتالي يُعاد وضع هذه المهام ضمن مخطط تدفق البيانات الجزئي G_2 عند إعادة تشكيله وفي مكانها المناسب.

وبالنتيجة نجد أننا لم نفقد أي عمل تم إنجازه قبل العطل وأنّ الاسترجاع كان محلّي للمعالج المعطّل فقط وأيضاً قد تم متابعة التنفيذ كما كان قبل العطل وكأنه لم يحدث.

4. النتائج ومناقشتها:

1.4. التعقيد الزمني للآلية المقترحة:

نناقش في هذه الفقرة الكلفة الزمنية لخوارزمية الآلية المقترحة لضمان صحة التشغيل، من أجل هذا الهدف نعرّف بعض المقادير المتعلقة بالتنفيذ المتوازي لتطبيق ما كما يلي:

t_s : الزمن الوسطي لتخزين مهمة ما على ذاكرة مستقرة.

N_{theft} : العدد الأعظمي للمهام المسروقة على معالج ما.

N_∞ : أكبر عدد للمهام المكونة للمسار الحرج (الجزء التسلسلي للتطبيق) في مخطط تدفق البيانات الممثل لتنفيذ التطبيق.

T_p : زمن تنفيذ التطبيق المتوازي بشكل طبيعي على P معالج.

T_p / τ : عدد نقاط الاستعادة الدورية على معالج ما من أجل دور τ.

1. **الكلفة خلال التنفيذ الطبيعي:** وتتضمن دراسة الكلفة المترتبة على استخدام الآلية المقترحة للتسامح مع الأعطال خلال التنفيذ الخالي من الأعطال.

إنّ الكلفة المضافة خلال التنفيذ الطبيعي تتمثل بكلفة أخذ نقاط استعادة دورية لكل معالج خلال دور مدته τ (يمكننا التحكم بطول الدور بحيث تكون حالة التطبيق صغيرة على المعالج عند أخذ نقطة الاستعادة) وكذلك كلفة تسجيل المهام المسروقة من كل معالج ضمن رتل العمل في الذاكرة المستقرة (يمكننا التحكم بمكان الذاكرة المستقرة فنجعلها قريبة).

بفرض أن T_p' يرمز إلى زمن تنفيذ البرنامج المتوازي على p معالج فيزيائي مع استخدام الآلية المقترحة، أي:

$$T_p' = T_p + CoD$$

حيث CoD هي كلفة الآلية المقترحة المضافة على البرنامج وهذه الكلفة تكون محدودة بأعظم كلفة لتسجيل نقاط الاستعادة الدورية لمعالج ما i $(i=1,\ldots,p)$ بالإضافة لكلفة تسجيل المهام المسروقة منه.

أما كلفة تسجيل نقاط الاستعادة الدورية لمعالج ما i فهي تعتمد على العدد الكلي لنقاط الاستعادة المحفوظة للمعالج i والكلفة الوسطية لتنفيذ نقطة استعادة واحدة.

- العدد الأعظمي لنقاط الاستعادة المحفوظة لمعالج ما i هو T_p'/τ وذلك من أجل الدور τ للمعالج (نختار هذا الدور بحيث يساوي الزمن الأعظمي لتنفيذ مهمة ما).
- أما كلفة نقطة الاستعادة الواحدة فهي تتعلق بعدد المهام في البيان الجزئي G_i في لحظة معينة ولما كانت خوارزمية التنفيذ المحلية التي نعتمدها في دراستنا على كل معالج تقوم على مبدأ التنفيذ بالعمق أولاً فإنّ عدد مهام البيان الجزئي G_i في لحظة ما يكون محدود بالمقدار N_∞ .

ومنه فإن كلفة نقاط الاستعادة الدورية في التطبيق محدودة بالمقدار:

$$(T_p'/\tau) * f_{overhead}(N_\infty, t_s)$$

حيث $f_{overhead}(N_\infty, t_s)$ دالة تتعلق بأكبر عدد للمهام المنفذة على المعالج في لحظة معينة وبالزمن اللازم لتخزين تلك المهام ضمن الذاكرة المستقرة.

وأما كلفة تسجيل عمليات السرقة ضمن رتل العمل في الذاكرة المستقرة تكون محدودة بأكبر عدد للسرقات وكلفة تسجيل تلك السرقات ضمن ذاكرة مستقرة أي يمكن التعبير عن ذلك بالمقدار : $N_{theft} * f_{overhead}(N_{theft}, t_s)$

حيث $f_{overhead}\ (N_{theft}, t_s)$ دالة تتعلق بأكبر عدد للمهام المسروقة وبالزمن اللازم لتخزين تلك المهام ضمن الذاكرة المستقرة.

ومنه فإنّ الكلفة الزمنية للتنفيذ المتوازي للتطبيق في مرحلة التخزين تعطى بالعلاقة:

$$T_p' \leq T_p + (T_p'/\tau) * f_{overhead}\ (N_\infty, t_s) + N_{theft} * f_{overhead}\ (N_{theft}, t_s)$$

وبحسب مبدأ العمل أولاً [13] فإنّ عدد السرقات يكون محدود وصغير في معظم التطبيقات وبالتالي باختيار مناسب للدور τ فيمكن أن نقول أن الكلفة المضافة التي حصلنا عليها نتيجة استخدام الآلية المقترحة للتسامح مع الأعطال صغيرة بقدر كاف.

2. **الكلفة عند حدوث عطل:** وتتضمن تقدير كلفة الخسائر الناتجة عند حدوث عطل ما أثناء التنفيذ والتي تتمثل بالكلفة الزمنية المرتبطة بالمهام المُعاد تنفيذها بعد العطل وذلك في ظل استخدام الآلية المقترحة للتسامح مع الأعطال.

إن الاسترجاع باستخدام الآلية المقترحة يتمثّل بإعادة بناء مخطط تدفق البيانات الخاص بالمعالج المعطّل من آخر نقطة استعادة محفوظة له بالإضافة لاستخدام رتل العمل من أجل استعادة عمليات السرقة وبالتالي فإن زمن تنفيذ المهام المفقودة نتيجة العطل وفق استراتيجية التخزين المقترحة يكون محدود بعدد مهام البيان الجزئي G_i وأن عدد المهام مرتبط باختيار الـ τ (دور أخذ نقاط الاستعادة للمعالج) ولمّا كان هذا الدور يساوي الوقت الأعظمي لتنفيذ مهمة ما فإن الخسارة تكون على الأكثر هي مهمة واحدة.

5. الاستنتاجات والتوصيات:

قدمنا في هذا البحث طريقة جديدة لضمان سلامة التشغيل باستخدام آلية جديدة للتسامح مع الأعطال للتطبيقات المتوازية التي تعمل في بيئات موزعة وواسعة وديناميكية وغير متجانسة؛ ترتكز على استخدام مخطط تدفق البيانات كنموذج مجرد لتمثيل تنفيذ التطبيق المتوازي وقد كان هذا الاختيار لعدة أسباب:

1) يمثل حالة عامة للنظام بشكل مستقل عن عدد الموارد المتوفرة.
2) مناسب لبيئة الحوسبة الشبكية الديناميكية كونه مخطط ديناميكي الإنشاء (يتكون خلال التنفيذ).
3) استُغلَّ هذا المخطط للتعامل مع صفة عدم تجانس المكونات المشاركة في بيئة الحوسبة الشبكية حيث يمثل حالة التطبيق بشكل مجرّد وبالتالي يمكن نقله وإعادة تنفيذه ضمن الموارد غير المتجانسة.

ولقد استخدمنا هذا المخطط حتى نقوم بعملية حفظ احتياطي لأجزاء سير التنفيذ بهدف إعادة تنفيذ المعلومات المحفوظة ضمن بيئة تنفيذ مختلفة في وقت لاحق عند ظهور حدث طارئ (عطل). وأخيراً يمكننا تحديد ميزات استخدام الآلية المقترحة بما يلي:

- الاسترجاع محلّي بالنسبة للمعالج المعطّل فقط.
- يكون التنفيذ بعد الاسترجاع حتمي، حيث يُعاد التنفيذ وكأنّ العطل لم يحدث.
- لن نجابه حادثة الدومينو (التنفيذ من البداية) لأنّ رتل العمل سيزوّدنا بجميع المعلومات الضرورية عن الاعتمادية بين المعالجات عند الاسترجاع بعد حدوث العطل.
- لن يظهر لدينا مهام أيتام (لا نعرف من أنشأها) لأن المهام التي يتم سرقتها يسجلها المعالج الذي ينفذ المهمة الأساسية ضمن الذاكرة المستقرة.

التوجه المستقبلي لهذا العمل سيكون على شكل مكتبة برمجية تضاف إلى بيئة البرمجة المتوازية Xkaapi [12] التي تعتبر بيئة برمجية متوافقة مع نموذج البرمجة والتنفيذ المستهدف في هذا البحث.

5. المراجع:

[1] PALMIERI F, PARDI S, 2010-Towards a federated Metropolitan Area Grid environment: The SCoPE network-aware infrastructure, in Future Generation Computer Systems Vol. 26, Issue 8, Pages: 1241-1256.

[2] Feitelson D.G,2005-The supercomputer industry in light of the top500 data Computing in Science Engineering,7(1):42-47.

[3] Oliner A. and Stearley J, 2007- What supercomputers say: A study of five system logs, In 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks,pages 575-584.

[4] Bosilca, G. Bouteiller, A. Cappello, F. Djilali, S. Fédak, G., 2002-Mpich-v :Toward a scalable fault tolerant mpi for volatile nodes. In SuperComputing, USA.

[5] Anstreicher, K. Brixius, N. Goux, J. and Linderoth, J. Solving. 2000-large quadratic assignment problems on computational grids.

[6] KAUR I, SINGH S, 2014-Detection of Crash Transient Failure during Job Scheduling using Replication Technique, in IJECS International Journal Of Engineering And Computer Science , Vol. 03 Issue 07 Pages 6904-6908.

[7] NEOCLEOUS K, DIKAIAKOS MD, 2006-Grid Reliability: A Study of failures on the EGEE Infrastructure, in Proceeding of the CoreGRID Integration Workshop, pages : 165-176 .

[8] Avizienis, A. Laprie, J-C. and Randell, B. 2004- Dependability and its threats - a taxonomy. In IFIP Congress Topical Sessions, pages 91-120.

[9] AVIZIENIS A, LAPRIE JC, and RANDALL B ,2001- Fundamental Concepts of Dependability, in University of Newcastle upon Tyne, Computing Science.

[10] BALA A, CHANA I, 2012-Fault tolerance-challenges, techniques and implementation in cloud computing, in IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 1.

[11] Gautier, T. Roch, JL. Wagner, F. 2007- Fine Grain Distributed Implementation of a Dataflow Language with Provable Performances, International Conference on Computational Science, pages: 593-600.

[12] Gautier, T. Le Mentec, F. Faucher, V. Raffin, B. 2013 - X-kaapi: A Multi Paradigm Runtime for Multicore Architectures. ICPP 2013 , pages: 728-735.

[13] FRIGO M, LEISERSON CE, and RANDALL KH, 1998- The implementation of the Cilk-5 multithreaded language, in Proc. ACM SIGPLAN conference on Programming language design and implementation, Pages 212 - 223.