مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (38) العدد (1) 2016
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (38) No. (1) 2016

تطوير التطبيقات الموزّعة الموثوقة والقابلة للتجزئة باستخدام Jgroup/ARM

الدكتور بشّار محمّد•

علي اسماعيل••

(تاريخ الإيداع 9 / 12 / 2015. قُبِل للنشر في 15/ 2 / 2016)

ملخّص

يتطلّب الاعتماد المتزايد على الأنظمة الشبكية في النشاطات اليومية تزويدها لخدمات متوفّرة وموثوقة. تزوّد Jgroup خدمة متوفّرة من خلال إنشائها نسخ (Replicas) متعددة من الخدمة نفسها وتوزيعها على أجهزة متعددة، بينما تحقّق الموثوقية من خلال سماحها لنسخ الخدمة بالحفاظ على الحالة المشتركة فيما بينها وتنسيق نشاطاتها باستخدام تقنية استدعاء الطريقة البعيدة (Remote Method Invocation). خلافاً لـ Jgroup، تستخدم JavaGroups تقنية تمرير الرسائل (Message Passing) لتحقيق التنسيق بين النسخ.

تقارن هذه المقالة بين أداءي استدعاء طريقة المجموعة في Jgroup بنوعيه الوحيد (anycast) والمتعدّد (multicast) واستدعاء الطريقة في JavaGroups بنوعيه طريقة الحصول على أول إجابة (GET_FIRST) وطريقة الحصول على جميع الإجابات (GET_ALL). تحسّن هذه المقالة أيضاً من أداء منصّة العمل ARM (Autonomous Replication Management) المدمجة مع Jgroup (Jgroup/ARM) لزيادة دعمها مع التسامح مع الخطأ؛ من خلال إيجاد حل أفضل لمعالجة مشكلة تعطّل كامل أعضاء نسخ الخدمة في تعاقب سريع.

تتميز الآلية الجديدة بقيام نسخة واحدة فقط (النسخة القائدة) بإرسال حدث التجديد بدلاً من قيام كل نسخ الخدمة بإرسال هذا الحدث؛ مع محافظتها على الزمن اللازم لاكتشاف حالة التعطّل من قبل مدير النسخ (Replication Manager).

تُظهر نتائج المقارنة بين Jgroup و JavaGroups تفوّق الثانية عند وجود نسخة خدمة واحدة، بينما يتفوّق أداء الاستدعاء في Jgroup على JavaGroups مع تزايد عدد نسخ الخدمة. تظهر النتائج أيضاً تزايد ملحوظ في زمن الاستدعاء في JavaGroups مع تزايد حجم المصفوفة الممررة إلى الطريقة المستدعاة. الأمر الذي يجعل JavaGroups غير مناسبة للتطبيقات التي تتطلب نقلاً لحجوم كبيرة من البيانات وعدداً كبيراً من المخدمات، بينما تعتبر Jgroup مناسبة لذلك.

تبين نتائج تقييم أداء الحل المقترح بأنّه يخفّض عدد أحداث التجديد المرسلة مقارنةً مع حل ميلينغ تصل في حدّها الأعظمي إلى **37.5%**، وتستغرق Jgroup/ARM الفترة الزمنية نفسها التي يتطلّبها الحل السابق لاكتشاف تعطّل المجموعة بكاملها.

الكلمات المفتاحية: التسامح مع الخطأ؛ استدعاء الطريقة البعيدة؛ إدارة النسخ والإصلاح؛ منصة عمل مجموعة الغرض الموزع Jgroup؛ نظام اتصالات المجموعة JavaGroups.

• مدرس في قسم البرمجيات ونظم المعلومات – كلية الهندسة المعلوماتية – جامعة تشرين – اللاذقية – سورية.

•• طالب دراسات عليا(ماجستير) – قسم النظم والشبكات الحاسوبية – كلية الهندسة المعلوماتية – جامعة تشرين – اللاذقية – سورية.

مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (38) العدد (1) 2016
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (38) No. (1) 2016

Development of Distributed Partitionable Reliable Applications using Jgroup/ARM

Dr.Bashar Mohammad•
Ali Esmaeel••

(Received 9 / 12 / 2015. Accepted 15 / 2 / 2016)

□ ABSTRACT □

The increasing reliance on network systems in day-to-day activities requires that they provide available and reliable services. Jgroup provides available service through creating multiple replicas of the same service on multiple devices. Jgroup achieves reliable service by maintaining the shared state between the replicas and coordinating their activities through Remote Method Invocation. Unlike Jgroup, JavaGroups uses message passing to implement coordination between the replicas.

In this paper, we compare Jgroup and JavaGroups for different Group Method Invocation modes. These modes are Anycast and Multicast in Jgroup, GET_FIRST and GET_ALL in JavaGroups.

This paper also improves the performance of ARM (Autonomous Replication Management) which is embedded with Jgroup (Jgroup/ARM) for supporting fault tolerance, through finding a new solution to handle group failure where all remaining replicas fail in rapid succession. In this new solution, only one replica (the group leader) issues renew events (IamAlive) periodically, instead of sending it by every replica in the group, with taking the same period to discover group failure by Replication Manager.

Results of Comparison show that JavaGroups is faster than Jgroup when a single replica is used, whereas Jgroup outperforms JavaGroups with increasing number of replicas. The invocation delay in JavaGroups increases noticeably with increasing the size of array passed into the invoked method which make JavaGroups unsuitable for applications which require exchanging big sizes of data and use large number of servers, whereas Jgroup is suitable for that.

Results show that the new proposal reduces the number of renew events to 37.5% at most, and Jgroup/ARM takes approximately the same period of time to discover group failure as in Meling solution.

KEYWORDS: Fault Tolerance; Remote Method Invocation; Replication and Recovery Management; Jgroup Object Group Platform; JavaGroups Group Communication System.

• Lecturer, Department of Software and Information Systems, Faculty of Information Engineering, Tishreen University, Lattakia, Syria.
•• Postgraduate Student, Department of Systems and Computer Networks, Faculty of Information Engineering, Tishreen University, Lattakia, Syria.

مقدّمـة:

يتشكّل النظام الموزّع (Distributed System) [1] من تعاون مجموعة من الأجهزة المستقلّة مع بعضها لتزويد تطبيق معيّن؛ بحيث تظهر إلى مستخدمي هذا التطبيق كنظام وحيد مترابط. يدعم النظام الموزّع قابلية التوسّع (Scalability) من خلال سماحه بإضافة أجهزة جديدة، كما يتفوّق على نظيره المركزي في تحقيقه التوافرية ودعمه التسامح مع الخطأ. حتى يتمكّن النظام الموزع من العمل بطريقة صحيحة؛ يتوجّب على التقنية المستخدمة في بنائه أن تسمح بالتفاعل بين مكونات التطبيق العاملة على أجهزة مختلفة عبر شبكة الاتصال.

تهدف هذه المقالة إلى مقارنة أداء استدعاء الطريقة بين إحدى تقنيات بناء الأنظمة الموزعة وهي Jgroup وتقنية أخرى وهي JavaGroups من أجل أنماط استدعاء متعددة، كما تهدف إلى تحسين أداء منصّة العمل ARM (Autonomous Replication Management) المدمجة مع Jgroup (Jgroup/ARM) لزيادة دعمها مع التسامح مع الخطأ، من خلال إيجاد حل أفضل لمعالجة مشكلة تعطّل كامل أعضاء مجموعة المخدم في تعاقب سريع.

تعتبر RMI (Remote Method Invocation) [2] إحدى التقنيات المستخدمة في بناء الأنظمة الموزعة، فهي تسمح لأغراض جافا المتوضّعة على أجهزة مختلفة بالتفاعل مع بعضها البعض كما لو أنها على جهاز واحد. يستدعي غرض جافا طرائق على غرض المخدّم (غرض مخصّص لتزويد مجموعة من الخدمات)، وتعالج RMI عملية تنظيم (Marshaling) تشمل تجميع وتغليف للبارامترات الممررة إلى الطريقة المستدعاة والقيم المعادة منها.

تقدّم لغة البرمجة جافا أيضاً تقنية أخرى وبمستوى عالي لبناء الأنظمة الموزعة تعتبر امتداداً لتقنية RMI؛ وهي Jini [3]. تزوّد هذه التقنية خدمة الاستعلام (Lookup)؛ وتنهي حاجة الزبون إلى معرفة مكان تواجد الخدمة من خلال تزويدها لآلية الاكتشاف (discovery mechanism). تسمح آلية الاكتشاف للخدمات بإيجاد مواضع خدمة الاستعلام وتسجيل وكلاء لها ضمنها، كما تسمح لطالبي الخدمة –الزبائن– بتحديد مواضع خدمة الاستعلام والاتصال معها للحصول على وكيل الخدمة المراد استخدامها. تساهم CORBA(Common Object Request Broker Architecture) [4] في بناء الأنظمة الموزّعة، من خلال سماحها للبرامج المطوّرة بلغات مختلفة (مثل Java، C++ ، C وغيرها)؛ المتنوّعة في تحقيقاتها (implementations)؛ والعاملة في مواقع مختلفة أن تتصل مع بعضها البعض بسهولة، كما لو أنها في موضع واحد.

تمثّل المنحى العام لزيادة توافرية الخدمة في الأنظمة الموزعة بإنشاء نسخ متعدّدة (مضاعفة Replication) من الخدمة نفسها وتوزيعها جغرافياً على عدة أجهزة، ولكن ذلك يتطلب دعماً من النظام لنمط التفاعل one-to-many. لا تزوّد RMI و CORBA هذا النمط من التفاعلات؛ في حين تزوّده أنظمة اتصالات المجموعة (Group Communication Systems). تستخدم هذه الأنظمة نموذج مجموعة الغرض (Object Group Pattern) [5]، يقوم هذا النموذج بتجميع مجموعة من أغراض المخدم منطقيّاً في مجموعة (group)، ويصبح بإمكان الزبون أن يتفاعل بشكل شفّاف مع هذه المجموعة كما لو أنّها مخدّم وحيد مفرد. ينتج استدعاء طريقة على مجموعة الغرض من خلال تنفيذ الطريقة من قبل واحد أو أكثر من المخدمات الأعضاء في هذه المجموعة.

تقارن هذه المقالة بين نظامين من أنظمة اتصالات المجموعة وهما: Jgroup [6] و JavaGroups [7] من حيث أداء استدعاء طريقة المجموعة، وذلك عن طريق قياس زمن التأخير اللازم لتنفيذ هذا الاستدعاء من أجل أنماط استدعاء متعددة، وهي: الوحيد(Anycast) والمتعدّد(Multicast) في Jgroup، طريقة الحصول على أول إجابة (GET_FIRST) وطريقة الحصول على جميع الإجابات (GET_ALL) في JavaGroups.

قدّمت Jgroup خدمة عضوية المجموعة القابلة للتجزئة [17]. تتبّع هذه الخدمة مسار التغيّرات في عضوية مجموعة المخدم (مغادرة أحد الأعضاء أو انضمام أعضاء جدد) لتزوّد كل عضو بمنظار (view) يحوي قائمة بالأعضاء الحاليين في المجموعة. تسمح هذه الخدمة باستمرارية توفّر الخدمة في جميع أجزاء الشبكة؛ وذلك بعد حدوث تجزئة في شبكة الاتصال تؤدي إلى انفصال مجموعة جزئية من الأجهزة عن غيرها. في حين تدعم أنظمة اتصالات المجموعة التي تعتمد أسلوب التجزئة الأساسي (primary partition) مثل ISIS استمرارية الخدمة في جزء واحد فقط يسمى الجزء الأساسي، وتتوقف النسخ المنفصلة عن هذا الجزء عن تزويد الخدمة. تعالج Jgroup أيضاً عملية العودة من التجزئة من خلال خدمة دمج الحالة. تعيد هذه الخدمة جميع نسخ الخدمة إلى حالة عامة متناسقة لتحقيق التوافرية، معالجةً بذلك التحديثات المتناقضة الممكن حدوثها على الحالة المشتركة لمجموعة المخدم خلال انفصال أعضائها في أجزاء متعددة. استبدلت Jgroup تقنية RMI بتقنية تمرير الرسائل المتبعة في أنظمة اتصالات المجموعة الأخرى مثل JavaGroups و Spread، فاعتمدت بذلك تقنية واحدة في جميع تفاعلاتها؛ سواء الداخلية لتحقيق التنسيق بين نسخ الخدمة؛ أو الخارجية اللازمة لاتصال الزبون مع مجموعة المخدم، مما يسهّل من عملية تطويرها. تستخدم Jgroup نموذج المجموعة المفتوح Open Group Communication System الذي لا يتوجّب فيه على الزبون الانضمام إلى مجموعة نسخ الخدمة حتى يتمكّن من الحصول على الخدمة، في حين يتوجّب على الزبون في JavaGroups القيام بذلك لاعتمادها نموذج المجموعة المغلق Closed Group Communication System. من هنا يبرز دور عملية المقارنة بين نظامين يعتمدان تقنيتين مختلفتين لتحقيق التنسيق بين نسخ الخدمة (تمرير الرسائل و RMI)، ويتخذان نموذجين مختلفين في تعاملهما مع زبون الخدمة (مغلق ومفتوح)، وذلك لتحديد أنواع التطبيقات الشبكية الملائمة لاستخدام كل منهما.

نظراً لديناميكة الشبكة؛ الناتجة عن انضمام مخدمات جديدة إلى الخدمة ومغادرة مخدمات أخرى أو الناتجة عن حدوث حالات تجزئة بسبب انقطاع في اتصال الشبكة؛ فإن تحقيق التسامح مع الخطأ (Fault-tolerance) في مثل هذه البيئات يتطلّب إدارة ذاتية لحالات التعطّل من قبل النظام نفسه؛ من حيث اكتشافه لهذه الأعطال وإصلاحها، بالإضافة إلى إدارة لعملية توزيع النسخ على الأجهزة في بيئة الهدف. لا تمتلك Jgroup آلية تسمح بهذه الإدارة الذاتية، لذلك تم تطويرها إلى منصة العمل Jgroup/ARM (Autonomous Replication Management) [8]. تسمح المنصة الجديدة بتوزيع نسخ المخدّم ذاتياً (دون تدخل بشري) على الأجهزة، وتحافظ على عدد محدد وثابت من النسخ في بيئة الهدف، فعند تعطّل إحداها؛ يكتشف النظام ذلك ويقوم بإنشاء نسخة بديلة. يهدف البحث إلى تطوير Jgroup/ARM من خلال إيجاد حلّ أفضل لمعالجة مشكلة تعطّل كامل أعضاء المجموعة (نسخ الخدمة) بتعاقب سريع. لقد تم التوصّل إلى تحسين آلية اكتشاف هذا النظام لحالة التعطّل هذه والتي تم اقتراحها مسبقاً من قبل الباحث ميلينغ [8]. تخفّض الآلية المقترحة حركة البيانات (traffic) الناتجة عن آلية تجديد الإيجار السابقة مع محافظتها على الزمن اللازم لاكتشاف حالة التعطّل، فهي تتميز بقيام نسخة واحدة فقط (النسخة القائدة) بإرسال حدث التجديد بدلاً من قيام كل نسخ الخدمة بإرسال هذا الحدث.

تشترك Jgroup/ARM في أهدافها مع منصّات عمل التسامح مع الخطأ الأخرى مثل Delta-4 [11]، AQuA [9]، و FT-CORBA [10]. ولكنّها تتميّز عن هذه المنصّات بالأسلوب الذي تتّبعه في إدارة التسامح مع الخطأ والمبني على مفهوم السياسات (Policies) التي تسمح بإدارة ومعالجة ذاتية للأعطال الحاصلة دون تدخل

بشري، ودعمها للتطبيقات المعنية بالتجزئة (partition awareness)، بالإضافة إلى اعتمادها تقنية RMI في جميع التفاعلات بين مكوّناتها.

تم تنظيم هذه المقالة على النحو التالي: تقدّم الفقرة 3 توضيحاً لمراحل وآلية استدعاء طريقة المجموعة في كل من Jgroup و JavaGroups. تصف الفقرة 4 آلية عمل Jgroup/ARM وتوضّح مشكلة تعطل كامل نسخ المجموعة في تعاقب سريع، بالإضافة إلى توضيح الحل الجديد المقترح لحل هذه المشكلة ومقارنته مع حل ميلينغ. تعرض الفقرة 5 نتائج المقارنة التي تم التوصل إليها بين أداءي Jgroup و JavaGroups وتقييماً لأداء الحل الجديد عن طريق طرح مجموعة من السيناريوهات وعرض النتائج التي تم التوصل إليها. تختتم الفقرة 6 هذه المقالة من خلال اقتراح مجموعة من التوصيات والأعمال المستقبلية.

منـهجيـة الـبـحـث:

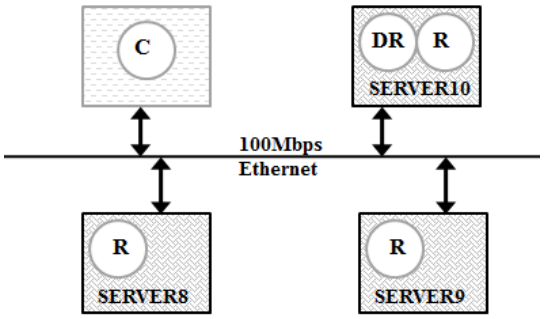
تمّ تحضير بيئة الهدف في مخبر الدراسات العليا في كلية الهندسة المعلوماتية بجامعة تشرين. تتألف هذه البيئة من مجموعة الأجهزة والبرمجيات المطلوبة لتنفيذ التجارب، حيث تمّ تحميل الأدوات الموضّحة في الجدول (1) واللازمة لتشغيل كلّ من تطبيقات Jgroup وتطبيقات JavaGroups وذلك بهدف إجراء عملية المقارنة.

جدول (1) البرمجيات المطلوبة للمقارنة بين أداءي Jgroup و JavaGroups.

Jgroup	JavaGroups
✓ Jgroup 3.0.1 distribution [6] ✓ Apache Ant version 1.7.0 [14]	✓ Jgroups 3.x distribution [12]
✓ Java J2SE version 1.6 [13] ✓ Apache Log4j version 1.3-alpha-8 [15]	

تم تطوير Jgroup باستخدام JAVA J2SE version1.5.، ويمكن تحميل هذه النسخة أو أية نسخة أحدث منها (هنا تم اختيار النسخة 1.6) واللازمة لعمل كلا النظامين. تزوّد الحزمة Apache Ant بنسختها 1.7.0 وسيلة سهلة وبسيطة لترجمة واختبار وتشغيل تطبيقات Jgroup. بينما تستخدم الأداة log4j في كلا النظامين بهدف تصحيح الأخطاء التي يمكن مواجهتها خلال تشغيل التطبيقات. كما تمّ تحميل الحزم البرمجيّة الخاصّة بمنصة العمل Jgroup/ARM على الأجهزة في بيئة الهدف، حيث تمّ إدخال التعديل المقترح على هذه المنصّة لتحسين آلية اكتشاف مدير النسخ لحالة تعطّل كامل نسخ الخدمة في تعاقب سريع، وإجراء تقييم الأداء اللازم لذلك.

تتألف بيئة الهدف الموضحة في الشكل(1) من أربعة أجهزة، تم تحميل ثلاث نسخ (Replicas R) على ثلاثة منها، وتم استضافة مسجل Jgroup (Dependable Registry) DR على المخدم SERVER10 ، أما العقدة الأخيرة فيتم من خلالها تشغيل برنامج الزبون وتنفيذ الاستدعاءات.



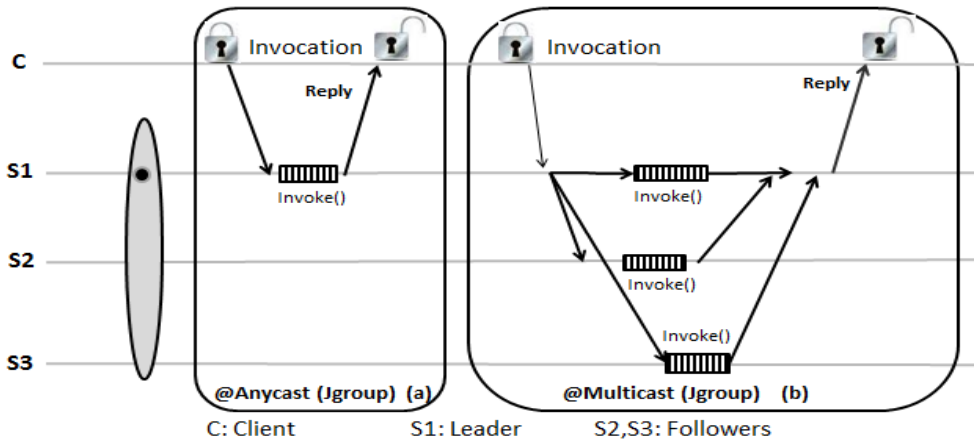
شكل 1: بيئة الهدف

بعد تحضير بيئة الهدف، تتم المقارنة بين أداءي استدعاء طريقة المجموعة في كلّ من نظامي Jgroup وJavaGroups عن طريق قياس زمن التأخير اللازم لتنفيذ أنماط الاستدعاء المختلفة (Anycast, multicast في Jgroup و GET_FIRST, GET_ALL في JavaGroups). يمثّل زمن التأخير في JavaGroups الفترة الفاصلة بين لحظة انضمام الزبون إلى مجموعة نسخ المخدم ولحظة حصوله على نتائج تنفيذ الاستدعاء، بينما يمثّل في Jgroup الفترة التي يستغرقها الوكيل الذي يحصل عليه الزبون في تنفيذه للاستدعاء والحصول على نتائج التنفيذ. تم طرح سيناريوهات متعدّدة، في السيناريو الأول يتم دراسة مدى تأثير حجم المصفوفة الممرّرة إلى الطريقة المستدعاة والتي تتراوح بين [0, 100KB] على زمن التأخير وذلك من أجل نسخة مخدّم واحدة فقط، أما في السيناريو الثاني تتم المقارنة من أجل نسختي مخدم وفي الثالث من أجل ثلاث نسخ خدمة. تعرض هذه المقالة أيضاً تقييم أداء الحل المقترح الذي يساهم في تخفيض عدد أحداث التجديد الدورية في منصة العمل ARM من خلال مقارنتها مع الحل المقدّم من قبل ميلينغ. يتمثّل معيار تقييم الأداء بالزمن الفاصل ما بين لحظة إستلام مدير النسخ لحدث تغير المنظار الأول (إنشاء أول نسخة) وحتى لحظة اكتشافه لحالة تعطّل كامل نسخ الخدمة، وفقاً لسلسلة معينة من أحداث التعطّل(الموضّح في الشكل 10)، حيث يتم تعطيل النسخ بنفس الطريقة قبل التعديل وبعده، ويتم حساب عدد أحداث التجديد المرسلة في كلا الحالتين (قبل التعديل وبعده).

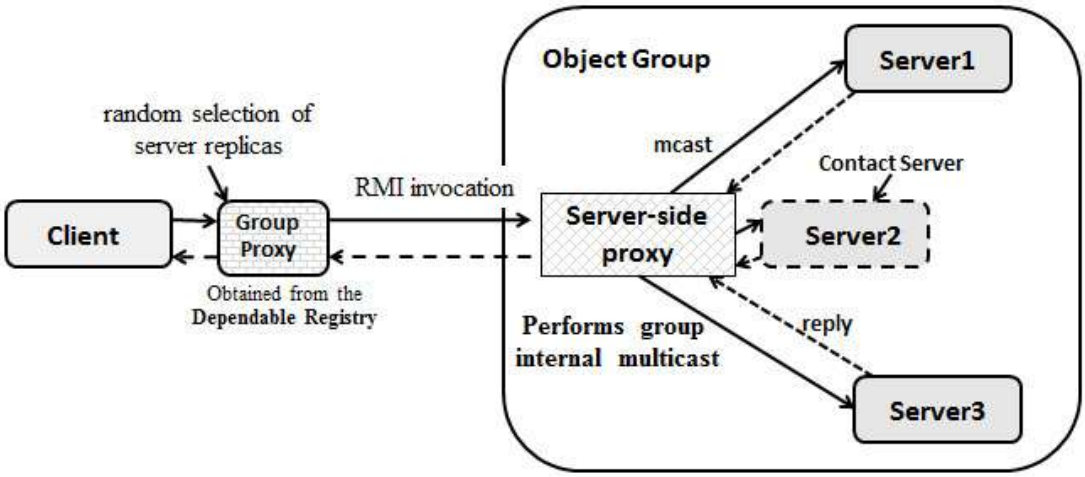
1 - آلية استدعاء طريقة المجموعة في كلّ من Jgroup وJavaGroups :

3-1- آلية استدعاء طريقة المجموعة في Jgroup:

يوضح الشكل2 نمطي الاستدعاء في Jgroup، في نمط الاستدعاء Anycast (الشكل 2 (a)) يتم إرسال الاستدعاء من قبل الزبون C إلى نسخة واحدة S1 فقط التي تنفّذ الاستدعاء ومن ثم تعيد الإجابة إلى الزبون. أمّا في نمط الاستدعاء Multicast (الشكل 2 (b)) يتم إرسال الاستدعاء بعد استلامه من قبل النسخة S1 إلى جميع النسخ الأخرى، ويتم التنفيذ على جميع هذه النسخ ومن ثم يتم إرسال إجابة واحدة فقط إلى الزبون.



الشكل 2: نمطي الاستدعاء في Jgroup، (a) نمط الاستدعاء Anycast (b) نمط الاستدعاء Multicast.

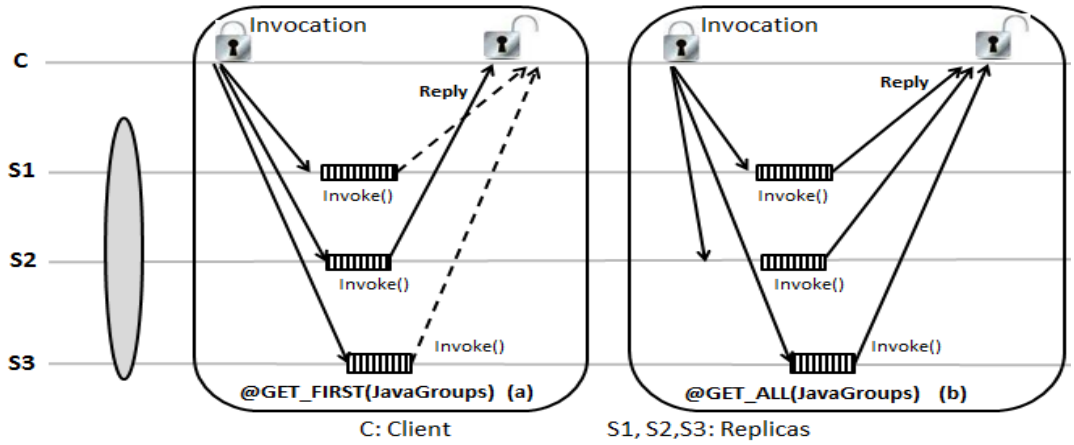


الشكل 3: مراحل تنفيذ استدعاء طريقة المجموعة في Jgroup.

يوضح الشكل(3) مراحل تنفيذ استدعاء طريقة المجموعة من النمط multicast. يحصل الزبون أولاً على وكيل المجموعة (group proxy)، ثم يستدعي الطريقة على هذا الوكيل. يختار الوكيل أحد المخدمات كممثّل له ضمن مجموعة الغرض (Server2 في الشكل3) ويوجّه الاستدعاء إلى الوكيل من جانب المخدم (server–side proxy). يكون وكيل المخدم المتصل به مسؤولاً عن إنجاز نمط الاستدعاء المطلوب، فمن أجل نمط البثّ المتعدّد يقوم بإرسال الاستدعاء إلى جميع الأعضاء ويجمع الإجابات الواردة، ويعيد استجابة وحيدة إلى الزبون المستدعي.

3-2- آلية استدعاء طريقة المجموعة في JavaGroups:

يوضّح الشكل4 نمطي الاستدعاء في JavaGroups. في النمط GET_FIRST (الشكل4 (a)) يتم قفل المستدعي حتى حصوله على أول إجابة من أحد الأعضاء، بينما يتم قفل الزبون المستدعي حتى حصوله على جميع الإجابات من جميع النسخ في النمط GET_ALL (الشكل4 (b)).



الشكل 4: نمطي الاستدعاء في JavaGroups، GET_FIRST (a)، GET_ALL (b).

ينشئ تطبيق الزبون في البداية صف القناة (Channel) والتي تعتبر النواة الأساسية في النظام [7]. ينضم من خلالها إلى مجموعة مخدم تملك نفس اسم القناة المنشأة. يتم استخدام الصف RpcDispatcher لتنفيذ استدعاء الطريقة المطلوبة، وذلك بعد تحديد نمط الاستدعاء المطلوب (GET_FIRST, GET_ALL).

تجدر الملاحظة هنا إلى عدم استخدام وكلاء في عملية الاستدعاء.

2 – آلية عمل Jgroup/ARM ومشكلة تعطّل كامل أعضاء المجموعة:

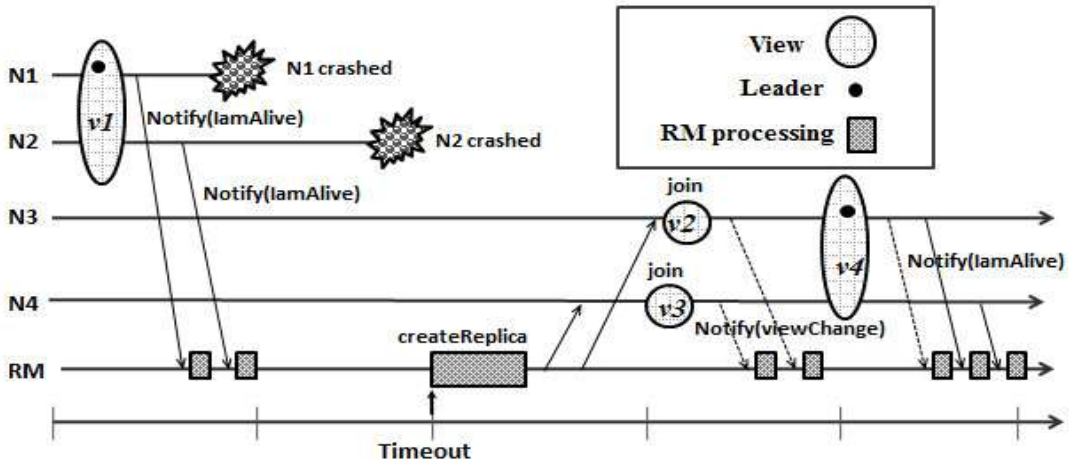
يتولّى المكوّن المركزي في Jgroup/ARM المسمّى مدير النسخ (Replication Manager RM) عملية توزيع النسخ على الأجهزة في بيئة الهدف، حيث يحدّد ملف السياسة المرتبط مع هذا المكون عدد النسخ المراد إنشاءها والذي يرمز له R_{init}. يجب أن يحافظ مدير النسخ على وجود عدد محدد من النسخ في بيئة الهدف، يعبّر عن هذا العدد ضمن ملف السياسة بالرمز R_{mini}، فإذا كانت قيمته تساوي 3 مثلاً، واكتشف مدير النسخ أن عدد النسخ 1 مثلاً، يتخذ مباشرة قراراً بإنشاء نسختين بديلتين محافظاً على عدد نسخ مساوياً للقيمة R_{mini}.

يجب على مدير النسخ مراقبة العضوية الحالية لمعرفة عدد النسخ المتوفرة وإجراء عملية الاستعادة المناسبة عند الضرورة. يحصل مدير النسخ على معلومات العضوية من خلال استلامه لأحداث تغيّر المنظار (veiwChange Event). تحمّل خدمة عضوية المجموعة في Jgroup منظاراً (view) على كل نسخة، يملك هذا المنظار حجماً يعبّر عن عدد الأعضاء ضمنه، وبما أن معلومات العضوية المتمثّلة بالمنظار متماثلة على جميع أعضائه، فإن إرسال إحدى هذه النسخ لتغيّر المنظار يعتبر كافياً لإبلاغ مدير النسخ بعضوية المجموعة الحالية. يتم إرسال حدث تغير المنظار من نسخة مميزة تسمى النسخة القائدة، وهي النسخة الأولى التي تم إنشاءها والتي تملك أصغر معرّف ID، وعند تعطّل النسخة القائدة يتم انتخاب النسخة التي تملك المعرف التالي. يبين الشكل 5 منظاراً(v1) مكوّناً من نسختين، حيث تم انتخاب النسخة التي تملك المعرّف الأصغر وهي N1 لتكون قائداً لهذا المنظار.

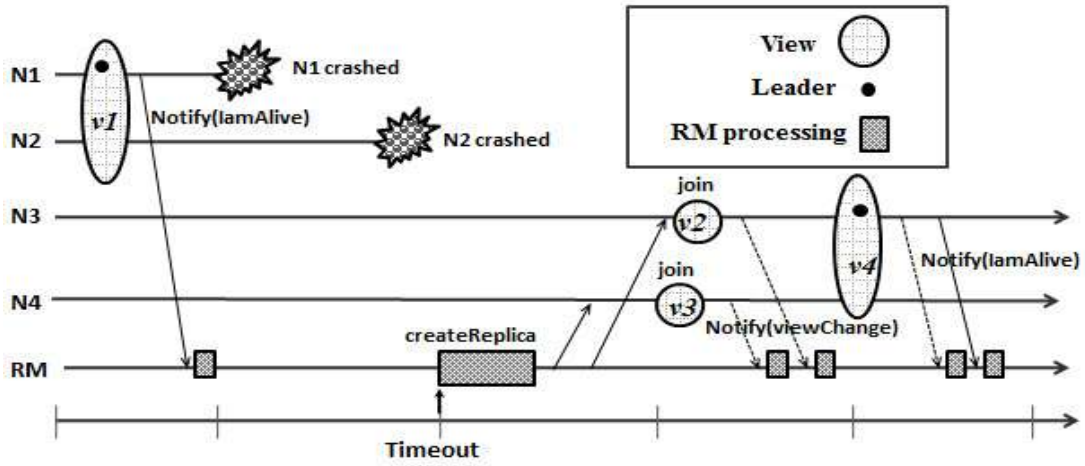
في بعض الحالات يمكن أن يحدث تعطّل لكامل نسخ المجموعة في تعاقب سريع دون أن تتبقى نسخة قائدة تقوم بإرسال حدث تغيّر منظار لإعلام مدير النسخ بحالة التعطّل هذه. لحلّ هذه المشكلة اقترح ميلينغ [8] إضافة آلية تجديد إيجار، تقوم فيها كلّ نسخة بإرسال حدث تجديد (IamAlive) وعلى فترات دورية متناسبة طردياً مع حجم مجموعة النسخ. فكلّما زاد حجم المجموعة ينخفض احتمال تعطّل كامل نسخها. عندما لا يستلم مدير النسخ أي حدث

تجديد، يكتشف حدوث تعطّل جميع نسخ الخدمة ويتخذ قراراً بإنشاء نسخ بديلة. في هذه الطريقة تزداد حركة البيانات في الشبكة مع تزايد عدد النسخ بسبب أحداث التجديد المرسلة من كافة نسخ الخدمة.

يميّز المنظار إحدى نسخه لتكون النسخة القائدة، وهي النسخة المسؤولة عن إرسال أحداث تغيّر المنظار عند حدوثها. إن غياب أية نسخة قائدة يعني غياباً للمنظار بأكمله (تعطّل المجموعة بكاملها)، فاحتواء المنظار لنسخة واحدة يقتضي اعتبارها النسخة القائدة، كما يتطلّب تعطّل النسخة القائدة في منظار انتخاب نسخة أخرى لتكون هي القائدة له. من هنا طوّرنا طريقة تقضي بقيام النسخة القائدة الحالية فقط بإرسال حدث التجديد بنفس الطريقة التي تقوم بها جميع النسخ في حل ميلينغ مع المحافظة على الزمن اللازم لمدير النسخ حتى يكتشف حالة التعطّل. وبالتالي يتم تخفيض عدد أحداث التجديد المرسلة في الحل السابق. يوضّح الشكل (5) طريقة ميلينغ لحلّ مشكلة تعطّل كامل نسخ المجموعة؛ حيث تقوم النسختان الموجودتان على العقدتين N1, N2 والمشكلتين للمنظار v1 بإرسال حدثي تجديد إلى مدير النسخ. عند انتهاء الفترة Timeout والتي تمثّل الفاصل الزمني المتوقع بين حدثي تجديد متتاليين؛ يكتشف مدير النسخ تعطّل النسختين(تعطّل المجموعة بكاملها) لعدم استلامه أي حدث تجديد، فيقوم بإنشاء نسخ بديلة على العقدتين N3, N4، يتم أخيراً تشكيل المنظار v4={N3, N4}، تقوم (N3) بدور القائد الذي يرسل حدث تغيّر منظار إلى مدير النسخ لإعلامه بالحالة الجديدة، ليتم بعد ذلك إرسال أحداث التجديد بنفس الطريقة. يتميز الحل الجديد الموضّح في الشكل(6) بإرسال النسخة القائدة فقط (N1 في حالة المنظار v1 و N3 في حالة المنظار v4) لحدث التجديد إلى مدير النسخ.



الشكل (5): اكتشاف تعطّل كامل أعضاء المجموعة بتعاقب سريع، وفقاً لحل ميلينغ.

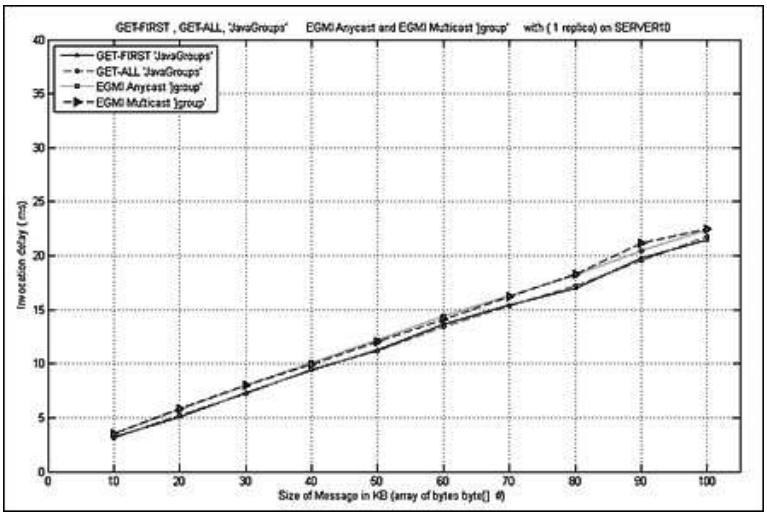


الشكل (6): اكتشاف تعطّل كامل نسخ المجموعة بتعاقب سريع وفقاً للحل المقترح.

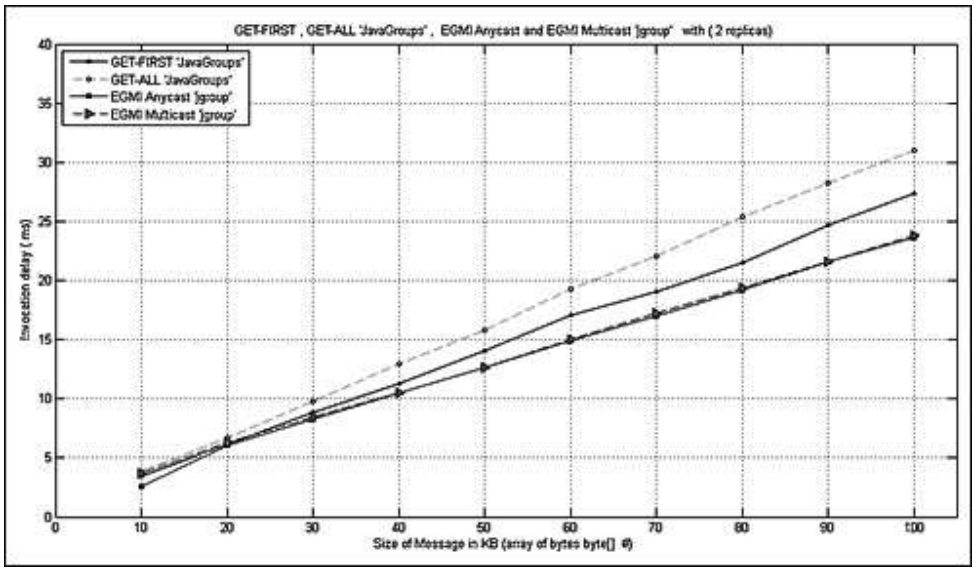
النتائج والمناقشة

مقارنة أداءي Jgroup وJavaGroups:

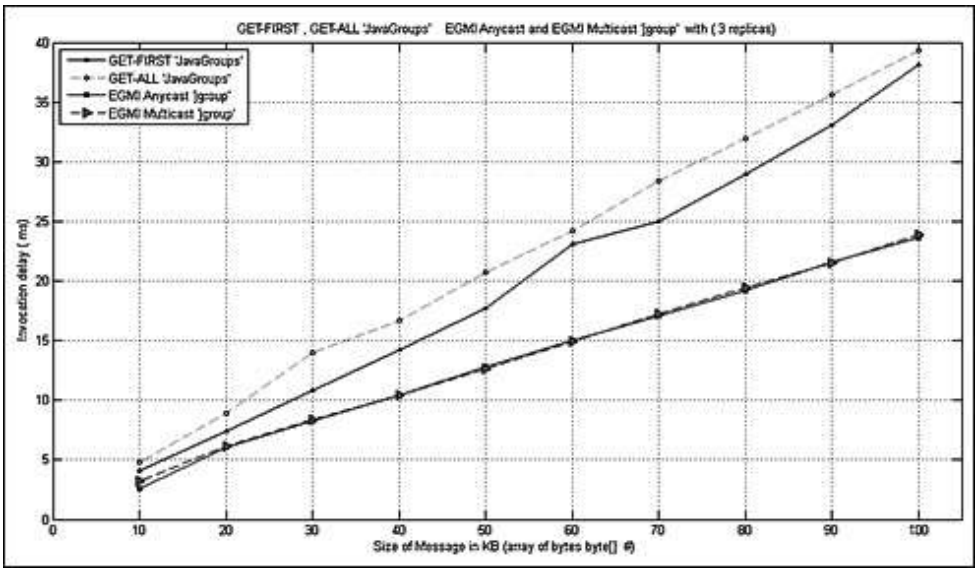
يمكن توضيح نتائج المقارنة من خلال المخططات التالية (الأشكال (7) (8) (9))



الشكل(7): تأثير حجم المصفوفة الممررة إلى الطريقة على زمن التأخير اللازم لتنفيذ الاستدعاء، مع وجود <u>نسخة مخدم واحدة فقط</u>.



الشكل (8): تأثير حجم المصفوفة الممررة إلى الطريقة على زمن التأخير اللازم لتنفيذ الاستدعاء، مع وجود نسختي مخدم.



الشكل (9): تأثير حجم المصفوفة الممررة إلى الطريقة على زمن التأخير اللازم لتنفيذ الاستدعاء، مع وجود ثلاث نسخ.

• يلاحظ من الشكل(7) تزايد زمن تنفيذ استدعاء طريقة المجموعة بشكل خطي مع تزايد حجم المصفوفة الممررة إلى الطريقة؛ وذلك في أنماط الاستدعاء الأربعة.

• في الشكل(7) حيث توجد نسخة خدمة واحدة فقط، تتفوّق JavaGroups في أدائها على Jgroup في أنماط الاستدعاء المختلفة، بينما تتفوّق تقنية Jgroup على JavaGroups في حالة نسختين وثلاث نسخ (الشكلين (8)(9))، ويمكن تعليل ذلك على النحو التالي:

✓ يقوم الزبون بعد انضمامه إلى مجموعة المخدم في JavaGroups ببثّ متعدد للاستدعاء إلى جميع نسخ المجموعة التي انضم إليها وينتظر حتى حصوله على أول إجابة (GET_FIRST) أو جميع الإجابات

(GET_ALL). لا تستغرق هذه العملية وقتاً طويلاً عند وجود نسخة مخدم واحدة فقط؛ في حين يتزايد زمن تنفيذ العملية بشكل ملحوظ مع تزايد عدد نسخ الخدمة بسبب الانضمام والإرسال إلى مجموعة بعدد نسخ أكبر. يوضّح الجدول (2) تزايد زمن التنفيذ مع تزايد عدد النسخ، وذلك من أجل مصفوفة ممرّرة بالحجم 100KB. في حين لم يتأثر زمن التنفيذ في Jgroup بتزايد عدد النسخ، حيث لم تتجاوز القيمة الفعلية للتأخير 25 ms، في حالة ثلاث نسخ وذلك في النمطين (Anycast, Multicast).

الجدول 2: أزمنة تنفيذ استدعاء طريقة المجموعة في نظام JavaGroups من أجل حجم مصفوفة 100 KB .

عدد النسخ N	التأخير أو الزمن اللازم لتنفيذ الاستدعاء Delay بالميلي ثانية	
	GET_FIRST	GET_ALL
1	21.466	21.793
2	27.347	30.982
3	38.111	39.328

✓ يتطلّب تنفيذ الاستدعاء في Jgroup معالجة ومروراً على طبقات أكثر نتيجة لاستخدام وكيل من جانب الزبون ووكيل من جانب مجموعة المخدم (الشكل 3). إن تنفيذ الاستدعاء على مجموعة مخدم مكونة من نسخة واحدة في Jgroup مع وجود الوكلاء يستغرق زمناً أطول من الزمن الذي يتطلّبه الزبون في JavaGroups ليستدعي الطريقة على مجموعة مكوّنة من نسخة واحدة. ولكن، مع تزايد عدد النسخ، يلاحظ تزايد الزمن اللازم لانضمام الزبون واستدعائه الطريقة في JavaGroups، في حين لا يتزايد هذا الزمن في Jgroup على الرغم من وجود الوكلاء، حيث يستمر الزبون بإرساله الاستدعاء إلى مخدم واحد يمثله ضمن مجموعة المخدم وينوب عنه في عملية الاستدعاء.

• يتزايد زمن تنفيذ الاستدعاء في نظام JavaGroups بشكل ملحوظ مع تزايد حجم المصفوفة الممررة، بينما لا نلاحظ هذا التزايد في Jgroup، يعود ذلك لاستخدام JavaGroups طبقة التجزئة FRAG2 والتي تقوم بتجزئة الرسالة المرسلة إلى عدة أجزاء، وعند تزايد حجم المصفوفة الممررة يتزايد عدد الرسائل المرسلة في كل استدعاء مما يزيد من زمن تنفيذه. لذلك لا تعتبر JavaGroups ملائمة للتطبيقات التي تتطلّب نقل حجوم كبيرة من البيانات كما في نقل الملفات.

• توضّح النتائج فروقاً أوضح بين النمطين GET-FIRST, GET-ALL في نظام JavaGroups منها بين النمطين EGMI Multicast, EGMI Anycast في تقنية Jgroup، حيث يكون الفرق بين زمني التنفيذ في النمطين (Anycast, Multicast) صغيراً جداً، وهذا يظهر مدى سرعة الوكيل من جانب المخدم المتصل به (الشكل(3) في إرساله الاستدعاء إلى باقي الأعضاء وحصوله على الإجابات في النمط Multicast. يكون فارق زمن التنفيذ بين النمطين anycast و multicast من أجل مصفوفة ممررة بحجم 80 bytes بحدود 0.3ms، في حين يكون بحدود 4ms في حالة النمطين GET_FIRST و GET_ALL من أجل حجم المصفوفة نفسه.

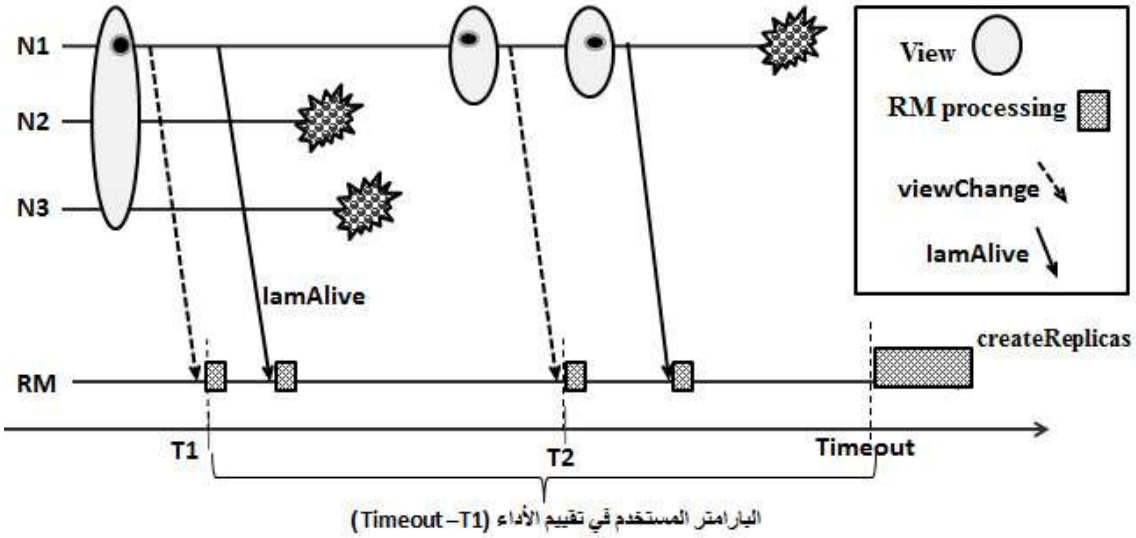
• **الخلاصة:** تعتبر Jgroup ملائمة للتطبيقات الشبكية التي تستخدم عدداً كبيراً من المخدّمات ونقلاً لحجوم كبيرة من البيانات، في حين تعتبر JavaGroups مناسبة للتطبيقات التي تستخدم عدداً قليلاً من المخدمات ونقلاً لحجوم صغيرة من البيانات، من الأمثلة عليها تطبيق Chat.

تقييم أداء الحل الجديد لمعالجة تعطّل كامل نسخ المجموعة بتعاقب سريع:

يتم طرح عدّة سيناريوهات لتقييم أداء الحل المقترح من خلال مقارنته مع حل ميلينغ. يوضّح الجدول 3 مجموعة من السيناريوهات المحتملة، بينما توضّح الجداول 4,5,6,7,8 النتائج التي تمّ التوصّل إليها في كلّ سيناريو، يمثّل R_{mini} عدد النسخ الأصغري المطلوب توفّره في بيئة الهدف، بينما يمثّل R_{init} عدد نسخ الخدمة الابتدائي التي يقوم مدير النسخ بإنشائها على الأجهزة في بيئة الهدف.

جدول 3 : السيناريوهات المحتملة

السيناريو	R_{init}	R_{mini}	توصيف السيناريو
1	3	1	تعطّل نسختين تابعتين، وبقاء النسخة القائدة فقط تقوم ببث حدث التجديد ومن ثم تعطّل هذه النسخة
2	3	1	تعطل نسخة تابعة ومن ثم تعطّل النسختين المتبقيتين (التابعة والقائدة) في تعاقب سريع
3	3	1	تعطّل جميع النسخ بتعاقب سريع
4	3	2	تعطّل نسخة تابعة، ومن ثم تعطّل النسختين المتبقيتين في تعاقب سريع
5	3	2	تعطّل النسخة القائدة ومن ثم تعطل النسختين المتبقيتين

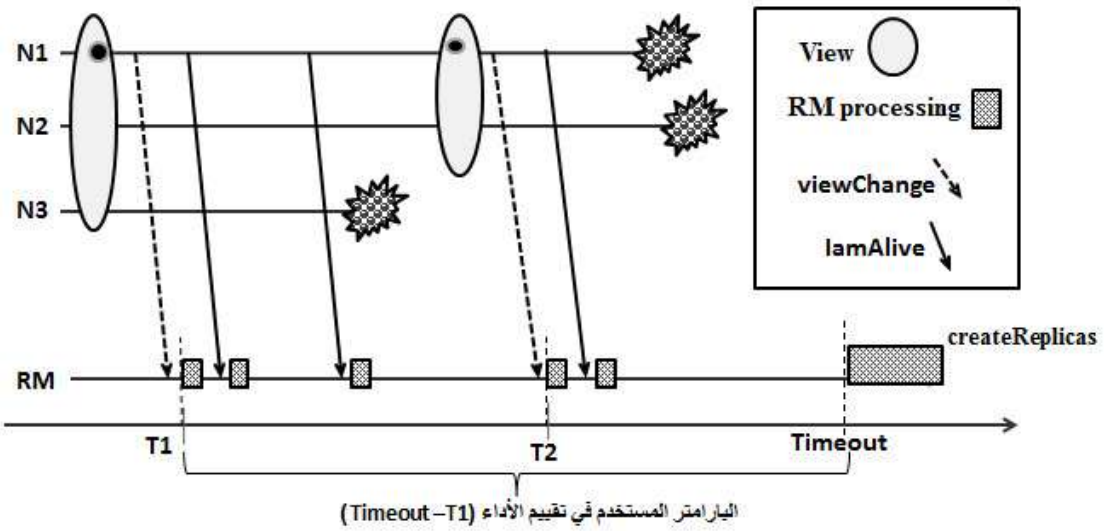


الشكل (10): سيناريو1، تعطّل نسختين تابعتين، وبقاء النسخة القائدة فقط تقوم ببث حدث التجديد ومن ثم تعطّل هذه النسخة.

يمثل الشكل(10) السيناريو الأول، حيث يكون لدينا منظار مكوّن من ثلاث نسخ متوضعة على العقد (N1,N2,N3). ترسل النسخة القائدة الموجودة على العقدة N1 حدث تغير المنظار ويستلمه مدير النسخ في اللحظة T1، ثم ترسل النسخة نفسها التجديد IamAlive. تتعطّل بعد ذلك النسختين التابعتين (Followers) الموجودتين على العقدتين N2,N3. ترسل النسخة القائدة N1 حدث تغير المنظار إلى RM، حيث يشير إلى الحالة الجديدة في العضوية وهي بقاء النسخة القائدة فقط. ترسل النسخة القائدة بعد ذلك حدث التجديد. بعد تعطّل النسخة القائدة، يكتشف مدير النسخ عند اللحظة Timout تعطل كامل النسخ لعدم استلامه حدث تجديد فيقوم بإنشاء نسخ بديلة.

جدول (4): نتائج سيناريو 1.

	عدد أحداث التجديد المرسلة قبل اكتشاف التعطّل.		الفاصل الزمني بين لحظة استلام أول تغير منظار(T1) ولحظة اكتشاف تعطّل المجموعة بكاملها(Timeout) مقدراً بالميلي ثانية		عدد مرات حدوث تغير حجم المنظار خلال كل تنفيذ	
	Meling	Ali	Meling	Ali	Meling	Ali
تنفيذ 1	4	2	63868	63577	4	4
تنفيذ 2	4	2	64159	63619	4	4
تنفيذ 3	4	2	63817	63039	4	4
تنفيذ 4	4	2	63852	64058	4	4
المتوسط			63924	63573.25		

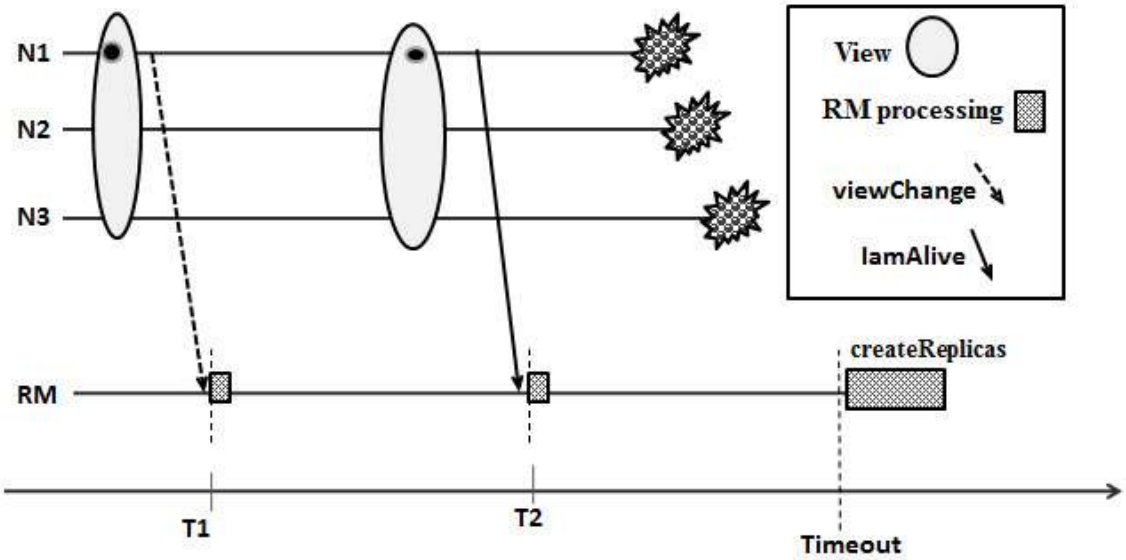


الشكل(11) : سيناريو2، تعطل نسخة تابعة ومن ثم تعطّل النسختين المتبقيتين (التابعة والقائدة) في تعاقب سريع .

يمثّل الشكل(11) السيناريو الثاني، حيث تقوم النسخة القائدة الموجودة على العقدة N1 بإرسال حدث تغير المنظار ليتم استلامه من قبل مدير النسخ عند اللحظة T1. ثم تقوم نفس النسخة بإرسال حدث التجديد على فترات دورية. تتعطل بعد ذلك النسخة التابعة الموجودة على العقدة N3. يؤدي ذلك إلى حدوث تغير في العضوية تبلغه النسخة القائدة N1 إلى مدير النسخ من خلال حدث تغير المنظار viewChange. تقوم النسخة القائدة بعدها بإرسال حدث التجديد. يعقب ذلك تعطل كلا النسختين N1,N2. عند انتهاء الفاصل الزمني المتوقع لاستلام حدث التجديد في اللحظة Timeout، يعلم مدير النسخ بحالة تعطل كامل النسخ ويتخذ قراراً بإنشاء نسخ جديدة على عقد أخرى.

جدول (5): نتائج سيناريو 2.

	عدد أحداث التجديد المرسلة قبل اكتشاف التعطّل.		الفاصل الزمني بين لحظة استلام أول تغير منظار(T1) ولحظة اكتشاف تعطّل المجموعة بكاملها(Timeout) مقدراً بالميلي ثانية		عدد مرات حدوث تغير حجم المنظار خلال كل تنفيذ	
	Meling	Ali	Meling	Ali	Meling	Ali
تنفيذ 1	8	3	63787	64122	4	4
تنفيذ 2	8	3	63950	64050	4	4
تنفيذ 3	8	3	63722	63875	4	4
تنفيذ 4	8	3	64116	63916	4	4
المتوسط			63893.75	63990.75		

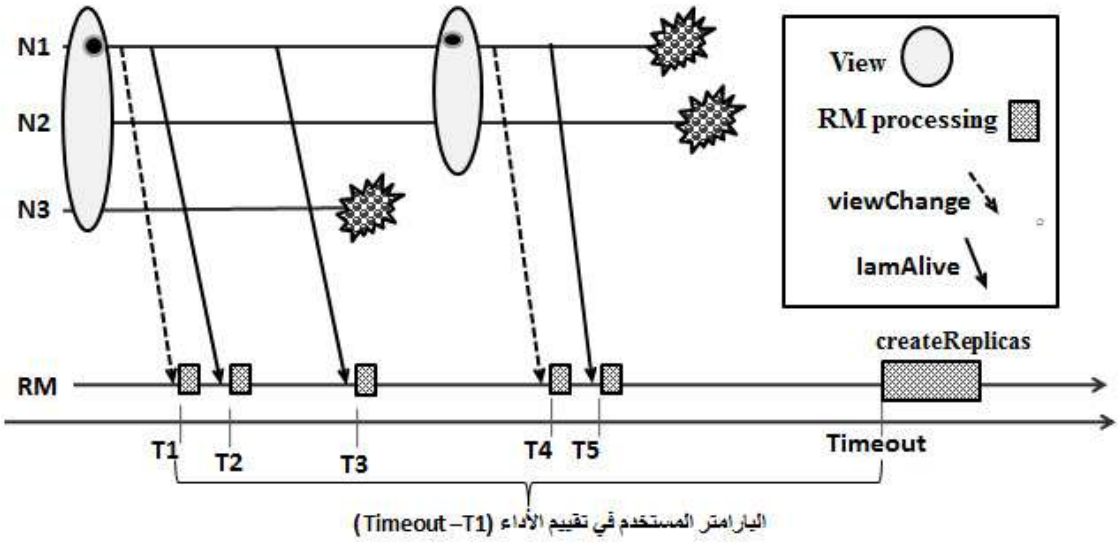


الشكل (12): سيناريو3 تعطّل جميع النسخ بتعاقب سريع.

يمثل الشكل (12) السيناريو الثالث. حيث تقوم النسخة القائدة بإرسال حدث تغير منظار إلى مدير النسخ ويتم استلامه عند اللحظة T1. يلي ذلك قيام النسخة القائدة N1 بإرسال حدث التجديد ليتم استلامه في اللحظة T2. تتعطل بعدها جميع النسخ بتعاقب سريع دون ارسال أية أحداث. عند انتهاء الفاصل الزمني المتوقع من مدير النسخ لاستلام حدث التجديد، وذلك عند اللحظة Timeout، يعلم بحالة التعطّل هذه ويتخذ قراراً بإنشاء نسخ بديلة على عقد أخرى.

جدول (6): نتائج سيناريو 3.

	عدد أحداث التجديد المرسلة قبل اكتشاف التعطّل.		الفاصل الزمني بين لحظة استلام أول تغير منظار(T1) ولحظة اكتشاف تعطّل المجموعة بكاملها(Timeout) مقدراً بالميلي ثانية		عدد مرات حدوث تغير حجم المنظار خلال كل تنفيذ	
	Meling	Ali	Meling	Ali	Meling	Ali
تنفيذ 1	3	1	82452	82139	3	3
تنفيذ 2	3	1	82279	82305	3	3
تنفيذ 3	3	1	81934	82252	3	3
تنفيذ 4	3	1	82046	81798	3	3
المتوسط			82177.75	82123.5		



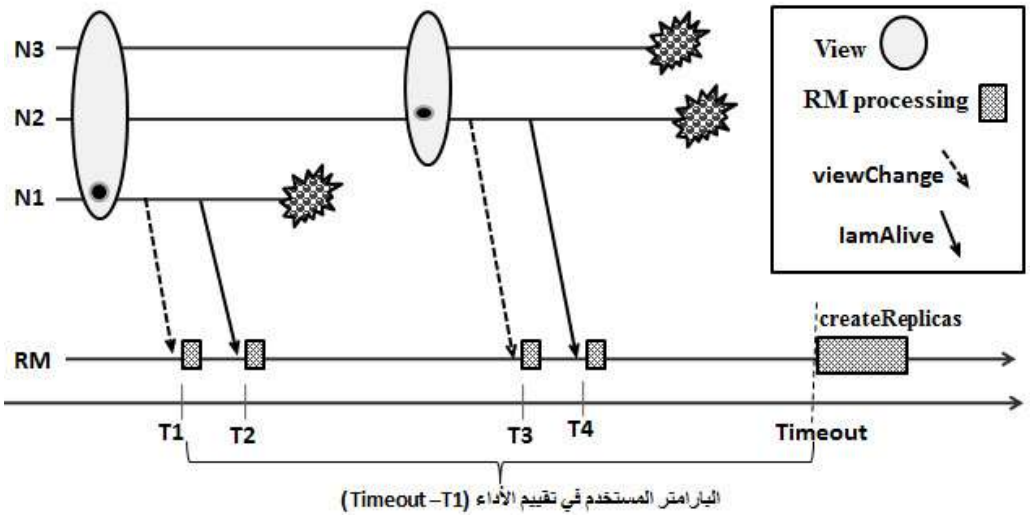
الشكل (13): سيناريو 4 تعطّل نسخة تابعة، ومن ثم تعطّل النسختين المتبقيتين في تعاقب سريع.

يمثل الشكل(13) السيناريو الرابع، حيث تقوم النسخة القائدة الموجودة على العقدة N1 بإرسال حدث التجديد على فواصل زمنية (يتم استلامها في اللحظات T2, T3). يتبع ذلك تعطّل النسخة التابعة الموجودة على العقدة N3.

يؤدي ذلك إلى حدوث تغير في العضوية يتم إبلاغه إلى مدير النسخ من خلال حدث تغير المنظار الذي ترسله النسخة القائدة N1. تقوم النسخة القائدة بإرسال حدث التجديد الذي يتم استلامه في اللحظة T5. يتبع ذلك تعطل كلا النسختين المتبقيتين: القائدة N1 والتابعة N2. عند انتهاء الفاصل الزمني المتوقع لاستلام حدث التجديد في اللحظة Timeout، يقوم مدير النسخ باتخاذ قرار لإنشاء نسخ بديلة عن جميع النسخ المتعطلة.

جدول (7): نتائج سيناريو 4.

	عدد أحداث التجديد المرسلة قبل اكتشاف التعطّل.		الفاصل الزمني بين لحظة استلام أول تغير منظار(T1) ولحظة اكتشاف تعطّل المجموعة بكاملها(Timeout) مقدراً بالميلي ثانية		عدد مرات حدوث تغير حجم المنظار خلال كل تنفيذ	
	Meling	Ali	Meling	Ali	Meling	Ali
تنفيذ 1	8	3	63991	63325	4	4
تنفيذ 2	8	3	64349	63863	4	4
تنفيذ 3	8	3	64269	63659	4	4
تنفيذ 4	8	3	63967	63831	4	4
المتوسط			64144	63669.5		



الشكل (14): سيناريو 5 تعطّل النسخة القائدة ومن ثم تعطّل النسختين المتبقيتين في تعاقب سريع.

يمثّل الشكل(14) السيناريو الخامس الممكن حدوثه. بعد إنشاء مجموعة نسخ ثلاثية، تقوم النسخة القائدة الموجودة على العقدة N1 بإرسال حدث تغير منظار يتم استلامه من قبل مدير النسخ في اللحظة T1. تقوم بعد ذلك

النسخة القائدة نفسها بإرسال حدث تجديد يتم استلامه في اللحظة T2 . يعقب ذلك تعطل النسخة القائدة. يؤدي تعطل النسخة القائدة إلى حدوث تغير في العضوية وانتخاب النسخة الموجودة على العقدة N2 لتصبح النسخة القائدة. تقوم النسخة القائدة الجديدة بإرسال حدث تغير المنظار إلى مدير النسخ الذي يتم استلامه في اللحظة T3. تقوم النسخة القائدة الجديدة بإرسال حدث التجديد، ليتم استلامه عند اللحظة T4 على مدير النسخ. تتعطل بعد ذلك النسختين N3,N2. لم يستلم مدير النسخ حدث التجديد المتوقع عند اللحظة Timeout، فيقوم بإنشاء نسخ بديلة عن النسخ المتعطلة.

جدول (8): نتائج سيناريو 5.

	عدد أحداث التجديد المرسلة قبل اكتشاف التعطّل.		الفاصل الزمني بين لحظة استلام أول تغير منظار(T1) ولحظة اكتشاف تعطّل المجموعة بكاملها(Timeout) مقدراً بالميلي ثانية		عدد مرات حدوث تغير حجم المنظار خلال كل تنفيذ	
	Meling	Ali	Meling	Ali	Meling	Ali
تنفيذ 1	5	2	62352	63156	4	4
تنفيذ 2	5	2	63137	62922	4	4
تنفيذ 3	5	2	63021	62702	4	4
تنفيذ 4	5	2	62340	62888	4	4
المتوسط			62712.5	62917		

الاستنتاجات والتوصيات :

- ✓ يخفّض الحلّ المقترح في Jgroup/ARM من حركة البيانات (traffic) المرسلة إلى مدير النسخ مقارنة مع حلّ Meling، وقد ظهرت فعالية هذا الحلّ في جميع سيناريوهات التعطّل المحتملة، وبشكل واضح في السيناريو 4 (الشكل 13)، حيث تم تخفيض عدد أحداث التجديد بنسبة 37.5% عن الحل السابق. يبدو تأثير الحل المقترح في تخفيض عدد أحداث التجديد جليّاً مع وجود نسخ خدمة بعدد أكبر، ففي السيناريو 3 مثلاً، وعند وجود 15 نسخة خدمة ضمن بيئة الهدف، تنخفض عدد أحداث التجديد من 15 في الحل السابق إلى حدث تجديد واحد في الحل المقترح.
- ✓ يحافظ الحلّ المقترح على الزمن اللازم لاكتشاف حالة تعطّل كامل نسخ الخدمة بتعاقب سريع، حيث أظهرت النتائج أن الفروق قليلة جداً (من رتبة الميلي ثانية) بين زمن الاكتشاف في الحلّ السابق وزمن الاكتشاف في الحلّ المقترح، ففي نتائج السيناريو 4 مثلاً (الجدول 7) ، بلغ متوسط أزمنة الاكتشاف للتنفيذات الأربع 64144ms في الحل السابق، بينما بلغ في الحل المقترح 63669.5ms.
- ✓ يساعد الحل المقترح على تحرير النسخ التابعة (Followers) من مهام إرسال أحداث التجديد، ويجعل مدير النسخ منشغلاً بانتظار ورود حدث تجديد واحد فقط بدلاً من انتظاره ورود عدد أكبر من هذه الاحداث والذي يزداد مع تزايد حجم المجموعة.

- ✓ تظهر نتائج المقارنة بين أداءي نظامي Jgroup, JavaGroups تزايد زمن تنفيذ استدعاء طريقة المجموعة بشكل خطي مع تزايد حجم المصفوفة الممررة إلى الطريقة وذلك في أنماط الاستدعاء الأربعة (Anycast, Multicast, GET_FIRST, GET_ALL).
- ✓ عند وجود نسخة خدمة واحدة فقط، تتفوّق JavaGroups في أدائها على Jgroup، بينما تتفوّق Jgroup على JavaGroups في حالة نسختين وثلاث نسخ من الخدمة.
- ✓ لا يؤثر عدد نسخ المخدم في Jgroup على زمن تنفيذ الاستدعاء في كلا النمطين multicast, Anycast. بينما في نظام JavaGroups يزداد زمن تنفيذ الطريقة بزيادة عدد نسخ المخدم.
- ✓ توضّح النتائج فروقاً أوضح بين النمطين GET-FIRST, GET-ALL في JavaGroups منها بين النمطين EGMI Multicast, EGMI Anycast في Jgroup. أما في Jgroup، فإن الفرق بين زمني التنفيذ في النمطين يكون صغيراً.
- ✓ يمكن مقارنة أداء Jgroup/ARM مع غيرها من الأنظمة الموزّعة المتسامحة مع الخطأ.
- ✓ من الممكن تحقيق دمج محكم بين مدير النسخ RM والمسجّل الموثوق DR في ARM، حيث توجد اعتمادية قويّة بين المكوّنين. إن عدم القدرة على تحقيق اتصال بين هذين المكوّنين نتيجة حدوث تجزئة في شبكة الاتصال يؤدي إلى فشل عمل النظام كاملاً، لذلك يتم وضعهما بشكل مشترك (co-located) على نفس الجهاز. يمكن استثمار الدمج المحكم بين RM وDR في حال التمكّن من تحقيقه لتحسين فعاليّة النظام كاملاً، حيث نحتاج بعد ذلك إلى قاعدة بيانات واحدة لتخزين المعلومات الحالية عن مجموعات الغرض. تعاني عملية الدمج هذه من مشكلة وحيدة، فلن يتم استخدام المسجّل الموثوق بدون مدير النسخ، وبالتالي سيتم تفعيل الاستعادة بشكل افتراضي في النظام.
- ✓ قام ميلينغ [16] بتطوير منصّة العمل Jgroup/ARM إلى منصة عمل أخرى (Jgroup/DARM) Distributed Autonomous Replication Management. تقوم DARM بتوزيع النسخ بطريقة موزّعة، حيث يتم الاستغناء عن المكوّن المركزي (مدير النسخ RM) الذي يحصل على معلومات عامّة عن كلّ مجموعات الغرض في النظام. تعالج كلّ مجموعة غرض في DARM حالات التعطّل بشكل ذاتي (self-healing) من خلال النسخة القائدة (Leader) لكلّ مجموعة. تعتبر دراسة DARM ومكوّناتها وآلية عملها بهدف تحسينها إحدى الأعمال المستقبليّة.

المـــراجـــع :

[1] Bernstein, P. *'Middleware: A Model for Distributed System Services' Communications of the ACM*, 39:2, 86–98. February 1996.

[2] Deitel, H.M., Deitel, P.J. and Santy, S.E. *Advanced Java 2 Platform: How To PROGRAM*. New Tersey: Prentice-Hall, Inc, 2002.pp. 854-905.

[3] Deitel, H.M., Deitel, P.J. and Santy, S.E. *Advanced Java 2 Platform: How To PROGRAM*, New Tersey: Prentice-Hall, Inc, 2002. pp. 1260-1317.

[4] Deitel, H.M., Deitel, P.J. and Santy, S.E. *Advanced Java 2 Platform: How To PROGRAM*, New Tersey: Prentice-Hall, Inc, 2002. pp. 1499-1571.

[5] Maffeis, S. *'The Object Group Pattern'*. In Proceedings of the 2nd USENIX Conference on Object-Oriented Technologies and Systems (COOTS), Toronto, Canada, June 1996.

[6] <http://Jgroup.sourceforge.net/index.html> [accessed in june 2013].

[7] Ban, B. *'JavaGroups:Group communication patterns in Java'*. Technical Report, Department of Computer Science, Cornell University, July 1998.

[8] Meling, H., Montresor, A., Helvik, B. E. and Babaoglu, O. *'Jgroup/ARM: a distributed object group platform with autonomous replication management'* .Softw. Pract. Exper., 38: 885–923. DOI: 10.1002/spe.853, 2008.

[9] Ren Y, Bakken DE, Courtney T, Cukier M, Karr DA, Rubel P, Sabnis C, Sanders WH, Schantz RE, Seri M. *'AQuA: An adaptive architecture that provides dependable distributed objects'*. IEEE Transactions on Computers 2003; 52(1):31–50.

[10] OMG. *Fault Tolerant CORBA Specification*. OMG Document ptc/00-04-04, Object Management Group, April 2000.

[11] Powell D. Distributed fault tolerance: *Lessons from Delta-4*. IEEE Micro 1994; 36–47.

[12] http://www.sourceforge.net/projects/javagroups/files/Jgroups/
[accessed in January 2013].

[13] <http://java-developmentkit-jdk.en.softonic.com/> [accessed in June 2012].

[14] <http://ant.apache.org/> [accessed in February 2012].

[15] <http://logging.apache.org/log4j/1.2/> [accessed in July 2012].

[16] Meling, H. *'An Architecture for Self-healing Autonomous Object Groups'*. University of Stavenger, Department of Electrical Engineering and Computer Science, N-4036 Stavenger, Norway, 2008.

[17] Montresor, A. *System Support for programming object-oriented dependable applications in partitionable systems (PhD Thesis*), University of Bologna, Department of Computer Science, February 2000. pp. 37-44.