یک رویکرد تحمل پذیر خطا برای عاملهای متحرک در سیستم FLASH

حجت اله حمیدی

دانشگاه آزاد اسلامی واحد دورود

E-mail:hojathamidi@ee.iust.ac.ir

چکیده- در طی سالهای گذشته بحث عامل (Agent)که ابتدا در قلمرو هوش مصنوعی مطرح شده امروزه در بسیاری از زمینه های کاربردی شناخته شده است. خصوصاً ، زیرگونه های عاملهای متحرک (Mobile Agent=MA) مورد توجه و علاقه فراوانی قرار دارند. علت این توجه نیز در میان دلایل دیگر، توانایی عامل در مهاجرت آسان در طول شبکه های متصل مانند اینترنت و یا سیستمهای کامپیوتری توزیع شده است. نشان داده شده که مهاجرت (حرکت) عامل بسیار سریعتر از مهاجرت پروسه های قدیمی است [6]. مشابه پروسه های قدیمی (سنتی) عاملها در یک سیستم توزیعی، با احتمال زیاد خطاهای سخت افزاری و نرم افزاری روبه رو هستند. تمایل به داشتن کاربردهایی با قابلیت اطمینان و یا قابلیت دسترسی بالا، اقدامات تحمل پذیری خطا را برای عاملهای متحرک ضروری می سازد. با این وجود پلت فرمهای عامل موجود، فاقد چنین اقداماتی هستند.در این مقاله ،افزودن عاملهای متحرک به سیستم FLASH با مکانیزم های تحمل پذیری خطا، شرح داده شده، که سر انجام به یک رویکرد تحمل پذیر خطا برای عاملهای متحرک(FANTOMAS= Fault-Tolerant Approach for Mobile Agents)منجر می شود.

کلید واژه- تحمل پذیر خطا ، گره ،FANTOMAS، FLASH ، Mobile Agent .

1- مقدمه

عامل متحرک (MA) یک هویت نرم افزاری فعال و مستقل است. هر MA خود تصمیم می گیرد که آیا روی گره کامپیوتری فعلی بماند یا به گره دیگری مهاجرت کند . در طول مهاجرت ، MA ، کد برنامه داده ها و در صورت امکان وضعیت اجرای خود را نگاه می دارد، پس از ورود به گره مقصد، MA به تحقق بخشیدن به یک کار(Task) مشخص ادامه می دهد، جهت انجام این کار عامل یا یک بخشی از کار را انجام می دهدو از سرویس های محلی استفاده می کند، یا با عاملهای محلی دیگر یا وسایل سرویس دهی تعامل ایجاد می کند. علیرغم استقلال عاملها آنها تنها نیستند و به صورت بالقوه با ایجاد تعامل سعی در رسیدن به اهداف مشترک (با دیگر عاملها) را دارند . علاوه براین، عاملهای متحرک قادرند که به اتفاقات و تغییرات مختلف واکنش نشان دهند. همچنین عاملهای متحرک به صورت فعال، قادر به درک محیط خود هستند و می توانند ابتکار عمل در به دست بگیرند[9] .

همه این قابلیتها، MAها را برای بسیاری از کاربردها در محیطهای توزیعی مانند جمع آوری اطلاعات جستجو و پالایش مدیریت شبکه ، تجارت الکترونیکی یا پردازش موازی [1,8] امکان پذیر می کند. کاربرد نمونه ای که ما

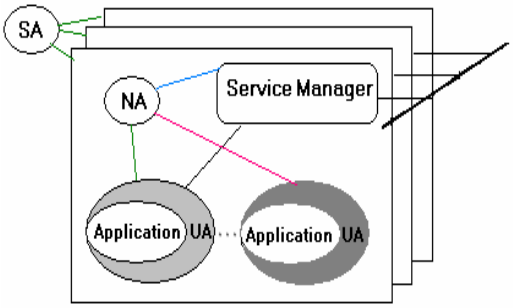
روی آن کار می کنیم نگاشت نمودارهای جریان داده ها برای کنترل روبات متحرک بر روی عاملها است [5].

بر خلاف کاربردهای (دیگر) که عموماً در آنها داده ها فقط می توانند به واحدهای پردازش انتقال داده شوند ،عاملهای متحرک عملگرها را به منبع داده ها می آورند. این کار می تواند بازده را افزایش داده و در عین حال (همزمان با آن) ترافیک شبکه را تا حد زیادی کاهش دهد[2] .

در حال حاضر سیستمهای عامل، که به طور رایج مورد استفاده می باشند، مستقیماً از عاملهای متحرک حمایت نمی کنند. در نتیجه بایستی سیستمهای عامل (Agent Systems) باید ایجاد شوند (رجوع به[4]) یک سیستم عامل، زیر ساخت اصلی برای ایجاد و اجرای عاملهای متحرک را عرضه می کند. با این وجود بسیاری از آنها به سرویس هایی همانند مدیریت بار یا تحمل پذیری خطای کاربردهای توزیعی اهمیت نمی دهند، به همین دلیل سیستم FLASH ، شناسایی و مورد استفاده قرار گرفته شده است.

FLASH -2

سیستم FLASH ، سیستم عامل انعطاف پذیر برای دسته ناهمگون می باشد، که یک ساختار مناسب برای ایجاد کاربردهای توزیعی با بار متعادل در محیطهای ناهمگون را عرضه می کند. جهت استفاده از FLASH ، برنامه نویسان قسمتهای موازی کاربردشان را توصیف کرده و آنها را توسط یک مکانیزم ظریف به عاملهای متحرک انتقال می دهند. عاملها آزادانه و به طور مستقل در طول یک دسته ناهمگون حرکت می کنند. و منابع مناسب را جستجو می کنند.



شکل 1: ساختار سیستم FLASH

در سطح عاملهای عمل کننده ، FLASH ، یک ساختار سلسله مراتبی دارد که برای اداره کردن سیستم مورد استفاده قرار می گیرد. در کل FLASH از چهار جزء اصلی، سه نوع عامل مختلف و یک مدیر سرویس تشکیل می شود (شکل 1) در راس سیستم یک یا چند عامل سیستم همکار (SA) ، گره های شرکت کننده گروه را مورد مشاهده قرار می دهند. در هر گره یک عامل گره ساکن (NA) قرار دارد که اطلاعات مربوط به گره، مدیر سرویس و عاملهای کاربر حاضر (UA) و غیره را نگهداری می کند، از آنجا که انتقال عاملهای کاربر به یک تعامل، برای بدست آوردن اطلاعات محلی نیاز دارد، عاملهای گره غیر متحرک (ساکن) هستند. با این وجود عاملهای گره، قادر به ایجاد پیکهای متحرکی هستند که بجای ایجاد کننده های خود می توانند عمل کنند. هر عامل کاربر حاوی بخشی از کاربرد موازی درون یک واحد کاربردی اضافی است .کار مورد نظر به عنوان کد برنامه قابل اجرا تعریف می شود، ولی کد می تواند از سرویسهای خارجی استفاده کند.

مدیر سرویس، سرویسهای سیستم فعال شده را تداوم می بخشد و از عاملهایی که از آن سرویسها استفاده می کنند، حمایت می کند. FLASH به برنامه نویسان اجازه می دهد

تا سیستم را با خدمات بیشتر توسعه دهند. بدین شکل کتابخانه ها و کاربردهای موجود می توانند توسط FLASH مورد استفاده قرار گیرند.

همه عاملهای اشاره شده به طور داخلی از چندین مدول تشکیل می شوند. در حال حاضر مدولهایی برای مهاجرت ارتباط و مدیریت بار وجود دارند. علاوه بر این عاملهای کاربر شامل یک فضای مشترک داده ای می شوند. مدول ها در زمان شروع عاملها، فعال می شوند و می توانند در طی زمان اجرا مبادله شوند.

تمام بخشهای FLASH که به عاملها تعلق دارند به زبان Java نوشته می شوند. علاوه بر این، سیستم Voyager [7] بر اساس مکانیزم های اصلی عاملها، یعنی مهاجرت و ارتباط را فراهم می سازد. سرویسهای سیستم مورد نظر می توانند در Java و همچنین ANSI C به کار برده شوند. بعنوان مثال سرویس اطلاعات منبع بومی مسئول ارزیابی بارسیستم روی هر گرهمی باشد.

مدول دیگری که در حال حاضر ایجاد می شود به یک رویکرد تحمل پذیر خطا برای عاملهای متحرک می پردازد که در بخش بعد توضیح داده می شود.

3- افزودن تحمل پذیری خطا به عاملهای متحرک

در ادامه مفاهیم عاملهای FLASH ، تحمل پذیری خطا یکی از ویژگی های مهم عامل متحرک است. برنامه نویس ،کاربر و حتی خود عامل تصمیم می گیرند که آیا و کجا از عامل استفاده کنند. مکانیزم تحمل پذیر خطا، باید با کاربرد ترکیب شود یعنی توسط یک یا چند مدول در درون عامل فراهم شود. این مکانیزم باید برای مقابله با پویایی بالای محیطهای عامل به اندازه کافی سریع باشد.

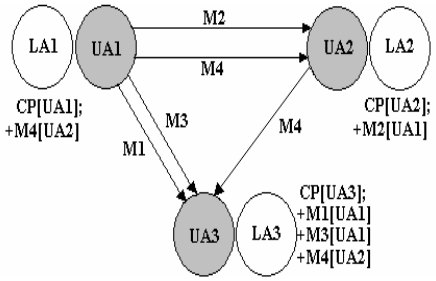
3-1- مدل خطا و تشخیص

یک کاربرد توزیعی و/ یا موازی از یک مجموعه عاملهای مستقل و همکار تشکیل می شود که روی گره های سیستم توزیعی اجرا می شوند. هدف فعلی کار FANTOMAS توانایی در مصون ماندن از خرابیهای تک عاملها یا تک گره ها است. فرض بر آن است که عاملها هرگز پیامهای اشتباه نفرستند. یک تک عامل می تواند به علت خطاهای سخت افزاری یا نرم افزاری از کار بیفتد. گره ها هم می توانند به علت مشکلات سخت افزاری یا نرم افزاری به طور موقت یا دائمی خراب شوند هنگامی که یک گره خراب شود، تمام عاملهایی که در آن لحظه روی آن گره اجرا می شوند نیز خراب می شوند.

در ادامه بحث عامل، تشخیص خرابی بایستی بصورت مستقل و با همکاری انجام شود. برنامه نویس، کاربرد می تواند با فراهم آوردن آزمونهای پذیرش، از عملیات صحیح عاملهای کاربر، اطمینان حاصل کند. عاملهای گره از کار افتادگیهای عاملهای کار بر محلی را آشکار می سازند، و از طریق پیامهای من زنده ام(I am alive messages) و وقفه ها گره های دورافتاده را کنترل می کنند.

3- نتیجه‌گیری

برای هر عامل کاربر (UA) ، یک عامل ثبت کننده اضافی (LA=Logger Agent) ایجاد می شود، تنها زمانی که تحمل پذیری خطا برای UA ضروری باشد (شکل 2) ، زمانی که UA مهاجرت می کند LA آن را با یک فاصله معین (برای مثال روی یک گره همجوار) تعقیب می کند. برای تحمل خرابیهای گرهی ،LA , UA هرگز نباید روی یک گره باشند.



شکل 2: مثالی برای یک کاربرد با 3 عامل کاربرد و ثبت کننده (Logger) با نقاط بازرسی (CP) و پیامها (M).

LA به عنوان یک نسخه ذخیره آماده برای UA عمل می کند. UA وضعیت خود را به LA اطلاع می دهد. پیامهایی که به UA فرستاده می شوند به LA نیز فرستاده می شوند. LA تمام پیامهایی را که پس از آخرین بازرسی نقطه ای دریافت شده اند ذخیره می کند. LA محاسبه کاربرد را اجرا نمی کند. UA یا LA (هردو) می توانند خراب شوند. اگر UA خراب شود LA آنرا دوباره با توجه به آخرین وضعیت دریافتی، می سازد و پیامهای ذخیره شده را دوباره در آن کپی می کند . اگر LA خراب شود UA یک LA جدید ایجاد می کند و آخرین وضعیت جاری را به آن می فرستد.

اضافه شدن عاملهای متحرک با این رویکرد حمایتی از طریق مکانیزم های متحرک موجود، تسهیل می شود. ایجاد LA از مهاجرت مشتق می شود . عامل هدف، کاربرد را ادامه نمی دهد، اما ثبت پیام را انجام می دهد و عامل مبدأ اجرای کاربرد را به پایان نمی رساند، اما آنرا ادامه می دهد. ایجاد دوباره یک UA خراب شده از نقطه بازرسی ذخیره شده در LA بهمین صورت عمل می کند.استراتژیهای تصمیم گیری: چه زمانی LA ایجاد شود/ مهاجرت کند، UA خراب شده در کجا مجدداً ایجاد شود و چه زمانی یک نقطه بازرسی فعلی از UA به LA (دوباره) ذخیره شود، می توانند به صورت مستقل و در همکاری با عاملهای دیگر، توسط .عاملها انجام شوند.معیارهای ممکن عبارتند از : بارگره، پایداری مشاهده شده توسط، آنها و رفتار ارتباطی کاربرد.

مراجع

[1] L.F. Bic, M.B. Dillencourt, M. Fukuda: Mobile Network Objects. To appear: J. Webster (Ed.), Encyclopedia of Electrical and Electronics Engineering, John Wiley & Sons, 1999.

[2] C.G. Harrison, D.M. Chess, A.Kershenbaum: Mobile agents: Are they a Good Idea?. Research Report, IBM T.J. Watson Center, 1995.

[3] M. Izatt, P. Chan, T. Brecht: Ajents: Towards an Environment for Parallel, Distributed and Mobile Java Applications. Proc. ACM 1999 Conf. on Java Grande, pp. 15-24, june 1999.

[4] J. Kiniry, D. Zimmerman: Special: A Hands-On Look at Java Mobile Agents. IEEE Internet Computing, 1/4, pp.21-30, 1997.

[5] R. Kluthe, W. Obel$\ddot{o}$ er, C. Grewe: Agent-based Load Balancing for Mobile Robot Application. To appear: Proc. Of the Int. Workshop on Distributed and Parallel Embedded Systems (DIPES'98). KLluwer Academic Press, 1999.

[6] D.S. Miloji$\check{c}ic'$, S. Buday, R. Wheeler: Old Wine in New Bottles – Applying OS Process Migration Technology to Mobile Agents. Proc. Of the 3rd ECOOP Workshop on Mobile Object Systems, 1998.

[7] ObjectSpace: Voyager Core Pachage – Version 3.0. Technical Overview, ObjectSpace, Inc., 1999.

[8] J. White: Mobile Agents White Paper. Technical Report, General Magic, Inc., Mountain View, 1996.

[9] M.J. Wooldridge, N.R. Jennings: Agent Theories, Architectures, and Languages: a Survey. Proc. ECAI-94 Workshop on Agent Theories, Arcgitectures, and Languages, LNAI 890, pp. 1-39, Springer, 1995.