Environnement d'exécution pour les applications réparties sous contraintes temporelles : une solution CORBA-RTP

Frank Singhoff
Isabelle Demeure

Ecole Nationale Supérieure des Télécommunications - CNRS URA 820
Département Informatique et Réseaux
46, rue Barrault, 75634 Paris Cedex 13
{singhoff, demeure}@inf.enst.fr

Résumé

Dans cet article, nous présentons une architecture orientée objet basée sur CORBA, qui permet l'exécution d'applications réparties possédant des contraintes temporelles. Elle exploite une technique qui, à partir d'une spécification de qualité de service, déduit automatiquement les échéances des tâches de l'application. Les protocoles RTP et RTCP sont utilisés pour la génération et le partage des informations d'ordonnancement.

Mots-clés : Ordonnancement temps réel, Qualité de service, CORBA, RTP

1. Introduction et motivations

Dans cet article, nous présentons l'environnement Polka. Il permet une prise en compte dynamique de contraintes de qualité de service (ou QoS pour Quality of Service) temporelles dans l'ordonnancement d'applications réparties. Le développeur ne précise aucune directive d'ordonnancement. Celles-ci sont automatiquement déduites par Polka à partir des contraintes de QoS spécifiées. L'ordonnancement ainsi mis en œuvre est orienté échéances. Dans la plate-forme décrite ici, l'ordonnanceur ne fournit pas de garantie sur le respect de ces échéances. En effet, il a pour cible les applications dont les besoins en ressources et le comportement ne peuvent être aisément déterminés. C'est notamment le cas de certaines applications multimédias où l'utilisateur ne peut pas aisément donner les temps d'exécution des différentes tâches du système (cf. [7]). Enfin, cette technique permet la construction d'applications portables et évolutives puisque l'adjonction de tâches supplémentaires dans une application, n'oblige pas le développeur à revoir le code de modules existants qui contiendraient des directives explicites d'ordonnancement. Une mise en œuvre de Polka pour des applications centralisées est décrite dans [6]. Cet article traitera plus particulièrement des mécanismes nécessaires au support des applications distribuées.

Le plan de cet article est le suivant : la section 2 définit l'architecture de Polka et le modèle utilisé pour spécifier les contraintes de QoS ; la section 3 explique comment déduire les échéances à partir des contraintes de QoS ; la section 4 présente les solutions proposées pour l'ordonnancement des applications Polka lorsqu'elles sont réparties ; enfin, la section 5 conclut en présentant nos travaux futurs.

2. Architecture du système Polka

Dans Polka, l'utilisateur modélise une application distribuée sous forme d'objets et d'équations de QoS (cf. [6]). Un objet est une unité d'encapsulation. Il peut être exécuté par un nombre quelconque d'activités (ou *threads*). Il peut exporter des interfaces définissant les méthodes susceptibles d'être invoquées par d'autres objets.

Notre plate-forme est construite autour d'un bus à objets CORBA[5]. CORBA est un standard défini par l'OMG. Ce standard spécifie un modèle d'objets distribués où les objets interagissent par invocation de méthodes. Le bus à objets fournit les fonctionnalités nécessaires à la réalisation de ces invocations, tout en assurant des propriétés d'interopérabilité et de transparence à la localisation. Les interfaces des objets sont spécifiées dans le langage de définition IDL (IDL pour Interface Definition Language).

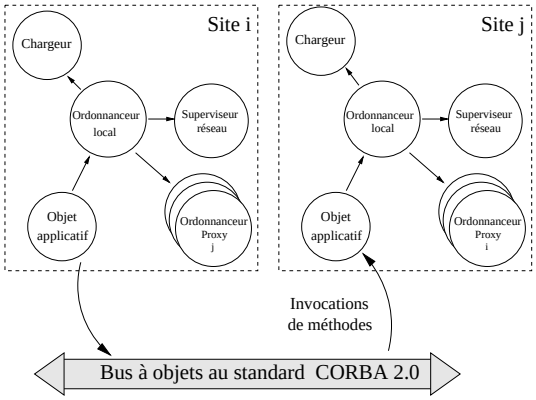

FIG. 1 – *Architecture du système* POLKA

Un système POLKA est donc composé d'une collection d'objets CORBA qui peuvent être placés sur des machines distinctes d'un réseau. Sur chaque site, en dehors des objets applicatifs, on trouve les objets suivants (cf. figure 1) : l' ordonnanceur local, le superviseur réseau, le chargeur, ainsi que plusieurs ordonnanceurs de proximité ou *proxies*. L'ordonnanceur local calcule les échéances à partir des équations de QoS. Il alloue le processeur aux tâches locales selon une politique EDF (Earliest Deadline First) [4]. Le superviseur réseau collecte des informations concernant l'état du réseau telles que le taux de perte des paquets et les délais de communication de bout en bout. Sur chaque site i, il existe un objet ordonnanceur *proxy* j pour chaque site j du système (avec $j \neq i$). Ces *proxies* offrent à l'ordonnanceur local une vue des informations d'ordonnancement manipulées par les objets ordonnanceurs distants. Ces informations sont nécessaires pour évaluer certaines échéances. Enfin, le chargeur initialise en mémoire les informations nécessaires à l'ordonnancement.

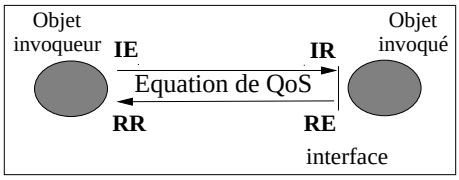

FIG. 2 – *Événements utilisés par* POLKA *durant une invocation de méthode*

Dans POLKA , les contraintes temporelles portent sur l'occurrence de certains événements. Ces événements correspondent à l'observation, en certains points, d'invocations de méthodes d'objets (cf. figure 2). Ce sont l'émission d'une invocation (événement IE) par l'initiateur ou client, la réception chez le serveur de cette invocation (événement IR), l'émission de la réponse qui fait suite (événement RE), et la réception de la réponse par le client (événement RR).

Les équations de QoS sont rédigées dans une logique temporelle : QL[2]. Ces équations expriment des contraintes d'échéance et de date d'exécution au plus tôt sur les événements cités précédemment.

Considérons, par exemple, l'équation :

$$\forall\ n\ :\ \tau(r.op1.RR, n) - \tau(r.op1.IE, n) \leq 10\ ms$$

$r.op1.IE$ et $r.op1.RR$ désignent les événements IE et RR observés lors de l'invocation de la méthode $op1$ de l'objet r. L'opérateur $\tau()$ donne la date des événements. Enfin, n est un entier désignant une occurrence particulière d'un événement (une méthode d'un objet peut être invoquée plusieurs fois, générant ainsi plusieurs occurrences). L'équation exprime qu'un délai maximum de 10 millisecondes doit être observé entre l'événement $r.op1.IE$ et l'événement $r.op1.RR$ correspondant.

3. Fonctionnement de l'objet ordonnanceur

L'ordonnanceur exploite deux informations: les équations de QoS et des graphes de tâches. Ces graphes sont obtenus par découpage du code de l'application. Il est réalisé en fonction des événements IE, IR, RE et RR ci-dessus mentionnés. Une tâche, qui est un nœud du graphe, est alors délimitée par deux événements. Ces tâches constituent des unités d'ordonnancement auxquelles sont associées une échéance et une date d'exécution au plus tôt. Les arcs de ces graphes expriment les dépendances entres les tâches.

Une activité exécute successivement les nœuds du graphe de tâches qui lui est associé. A toute activité éligible correspond une échéance qui est celle du prochain nœud à exécuter (dit nœud courant).

Chaque fois qu'il est appelé, l'ordonnanceur doit choisir l'activité à exécuter en fonction de l'échéance de son nœud courant. L'ordonnanceur est activé quand l'exécution d'une tâche se termine. Il doit alors effectuer les calculs nécessaires pour mettre à jour les échéances des graphes, puis élire l'activité dont le nœud courant possède l'échéance la plus proche.

Le principe de base pour la mise à jour des échéances est simple. Soit une équation $\tau(e, n+k) - \tau(e', n) \leq l$, où e et e' sont des événements, l un délai et k un entier. Si l'ordonnanceur dispose de $\tau(e', n)$, alors, l'échéance de l'occurrence $n+k$ de l'événement e peut être déterminée par $d_e = min(d_e, \tau(e', n) + l)$ où d_e est l'échéance de l'événement e. L'opérateur $min()$ est ici utilisé pour prendre en compte des valeurs de d_e calculées antérieurement.

Toutefois, ce calcul simple pose problème lorsqu'il est appliqué à des graphes, puisqu'il est alors possible de trouver des cas où un nœud possède des nœuds successeurs avec une échéance plus petite que la sienne. La solution proposée est d'évaluer l'échéance d_i du nœud i par la formule $d_i = min(d_i, min(d_j \mid i \prec j))$, où $\prec$ définit une relation de précédence telle que $i \prec j$ signifie que le nœud j ne peut pas être exécuté avant que l'exécution du nœud i soit terminée[1].

Cette solution résout également le cas des nœuds qui n'ont pas d'échéance (cas des événements n'apparaissant pas dans une équation de QoS). Toutefois, elle est conçue pour un ordonnancement monoprocesseur et doit être étendue pour un système réparti par la prise en compte des délais de communication. En effet, dans ce dernier cas, les occurrences de certains événements (tels qu'un IE et un IR), sont séparés par une ou plusieurs communications si les deux objets en interaction sont placés sur des machines différentes (ex : une communication est nécessaire entre un IE et l'IR correspondant). Il faut alors utiliser la formule $d_i = min(d_i, min(d_j - r_{i,j}.C_{i,j} \mid i \prec j))$ où $C_{i,j}$ est le temps de communication séparant les événements i et j et $r_{i,j}$, le nombre de communications réalisées entre les événements i et j.

En pratique, l'architecture décrite dans la figure 1 est complétée par un ensemble d'outils de compilation. Ils ont pour but l'analyse du code des objets et des contraintes de QoS, ainsi que la construction des graphes de tâches et la génération des souches et des squelettes CORBA (les souches et squelettes sont les composants logiciels qui mettent en œuvre les invocations à distance). Lors de la compilation d'une application, les souches et squelettes CORBA sont instrumentés pour appeler l'ordonnanceur à chaque occurrence des événements IE, IR, RE et RR.

4. L'objet de supervision réseau et les ordonnanceurs *proxies*

Nous détaillons ici le fonctionnement des ordonnanceurs de proximité et du superviseur réseau. Par la suite, nous supposerons que l'infra-structure de communication ne fournit pas de temps de communication déterministe. La solution présentée ci-après trouve alors, dans des réseaux tels qu'Internet, un contexte d'utilisation idéal.

Nous utilisons le protocole RTP[3] sur UDP. Il a pour objet le transport de données ayant des propriétés temporelles (données multimédias par exemple). RTP ne garantit pas le respect des propriétés temporelles : c'est au réseau sous-jacent que revient cette tâche. Une session RTP est constituée d'un ensemble d'extrémités qui émettent ou reçoivent un flot de données. Une session peut être unicast ou multicast. Enfin, à chaque session RTP est associé un flot de contrôle géré par le protocole de contrôle RTCP. Ce flot fournit, en particulier, une estimation du temps de communication des paquets RTP.

Dans notre architecture, une session RTP est ouverte pour chaque couple de sites du système. Contrairement au bus à objets, ce deuxième canal de communication ne véhicule pas de données des objets applicatifs. Il est utilisé pour l'échange des dates de fin d'exécution des tâches, qui sont nécessaires au calcul d'échéance. Les objets *proxies* mémorisent les dates véhiculées par les paquets RTP. Le superviseur réseau utilise RTCP pour obtenir une évaluation des temps de communication des paquets RTP. Il

maintient aussi une estimation des temps de communication des invocations de méthodes d'objets applicatifs : si $T1$ désigne la date d'un événement IE observé sur le site $S1$ et si $T2$ correspond à la date de la première action correspondante qui peut être observée sur le site $S2$, alors $T2 - T1$ correspond au temps nécessaire pour acheminer l'invocation du site $S1$ vers le site $S2$.

Nous montrons maintenant comment en exploitant les informations transportées dans un paquet RTP, nous pouvons éviter l'utilisation d'une horloge globale pour le partage des informations d'ordonnancement entre les différents ordonnanceurs locaux. Chaque paquet RTP contient la date de son émission : il est donc possible de "convertir" une date de fin d'exécution d'une tâche exprimée par rapport à l'horloge du site émetteur, en une date correspondante pour l'horloge du récepteur. Supposons que e soit l'événement associé à la fin de l'exécution d'un nœud sur le site i. Notons alors $\tau(i, e)$ la date de cet événement exprimée dans l'horloge de i. Cette information est émise dans un paquet RTP à la date $\tau(i, s)$, où s est l'événement associé à l'émission du paquet. Le paquet RTP contient alors, à la fois $\tau(i, e)$ et $\tau(i, s)$. Ce dernier arrive sur le site j à la date $\tau(j, r)$ où r est l'événement associé à la réception du paquet RTP. Puisque RTCP fournit une estimation du temps de transmission de ce paquet, le site récepteur peut "convertir" la date $\tau(i, e)$ en $\tau(j, e)$ par le calcul : $\tau(j, e) = \tau(j, r) - C - (\tau(i, s) - \tau(i, e))$ où C est l'estimation du temps de communication d'un paquet RTP et $\tau(i, s) - \tau(i, e)$ le délai entre l'événement e et l'émission du paquet RTP correspondant.

5. Travaux futurs

Dans cet article, nous avons expliqué comment on pouvait déduire l'ordonnancement d'une application à partir d'une spécification de contraintes de QoS. Aujourd'hui, la plate-forme décrite dans la section 2 est partiellement implantée. La version actuelle supporte, en effet, uniquement les applications centralisées. Cette première étape a pu être validée par le développement d'une application multimédia manipulant une quantité importante de données vidéo et audio et où une réservation de ressource "au pire cas" n'était pas envisageable. Cette application a démontré la flexibilité de POLKA et sa capacité à prendre en compte la QoS spécifiée. L'intégration de RTP et la modification de notre ordonnanceur afin de prendre en compte les temps de communication, nous permettra de tester l'intégralité de la solution proposée.

Enfin, cet article montre comment POLKA peut ordonnancer des applications partiellement non prédictibles. Cependant, si les temps de communication sont déterministes et si les temps d'exécution des nœuds sont fournis, il est possible d'appliquer les techniques décrites ici à des applications nécessitant un respect strict des contraintes temporelles. Dans cette optique, nous étudions actuellement les conditions permettant de décider de l'ordonnançabilité de telles applications.

6. Remerciements

Ce travail est réalisé dans le cadre d'une collaboration avec l'équipe de Jean-Bernard Stefani au Centre National d'Études des Télécommunications (CNET).

Bibliographie

1. J. Blazewicz. Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines. In. Gelende. H. Beilner (eds), Modeling and Performance Evaluation of Computer Systems, Amsterdam, Noth-Holland, 1976.
2. J.B. Stefani G.S. Blair G. Coulson M. Papathomas P. Robin F. Horn L. Hazard. A programming model and system infrastructure for real-time synchronization in distributed multimedia systems. *IEEE Journal on selected areas in communications*, 14(1), January 1996.
3. H. Schulzrinne S. Casner R. Frederick V. Jacobson. RFC1889 : RTP : A Transport Protocol for Real-Time Applications. Network Working Group, pages 1-75, January 1996.
4. C. L. Liu J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.
5. OMG TC Document 97-09-01. The Common Object Request Broker : Architecture and Specification. Revision 2.1. September 1997.
6. I. Demeure F. Singhoff. Spécification et ordonnancement dynamique d'applications multimédias : l'environnement POLKA . pages 101–117. Real time systems'98, Paris la Défense, janvier 1998.
7. V. Baiceanu C. Cowan D. McNamee C. Pu J. Walpole. Multimedia Applications Require Adaptive CPU Scheduling. Workshop on Resource Allocation Problems in Multimedia Systems, Washington DC, December 1996.