کاربرد الگوریتمهای خوشه بندی در تشخیص توزیع بهینه برنامه های محاسباتی شئ گرا در سطح شبکه های همگون اختصاصی

سعید پارسا، امید بوشهریان

چکیده

در این مقاله نشان داده شده که معیارهای حداکثر چسبندگی و حداقل اتصال بین قطعات معماری در برنامه های محاسباتی توزیعی معیار صحیحی نمی تواند برای ارزیابی معماری باشد. چرا که در این نوع برنامه های توزیعی هدف افزایش سرعت اجرایی با ایجاد حداکثر همروندی در اجرای قطعه های توزیع شده است. در این راستا در این مقاله یک رابطه کلی تخمین زمانی به عنوان تابع هدف جهت سنجش ارزش توزیعهای متفاوت از کد برنامه های شیی گرای محاسباتی ارایه شده است. این رابطه تخمین زمانی که با پیمایش گراف جریان فراخوانیها ایجاد می شود، برای هر فراخوانی دو حالت غیر همگام راه دور و ترتیبی را در نظر می گیرد. نوع فراخوانیها بر اساس خوشه بندی مدل ارتباطی کلاسهای استخراج شده از برنامه مورد نظر مشخص می شود. هدف، بدست آوردن خوشه بندی ای است که حداکثر همروندی را در اجرای برنامه حاصل نماید. با این روش می توان بهترین ترکیب توزیع و تعداد ایستگاه های کاری برای توزیع برنامه ها در سطح شبکه را بدست آورد.

کلمات کلیدی

محاسبات توزیع شده، الگوریتمهای خوشه بندی، تخمین زمانی، پیمانه سازی

The Application of Clustering Algorithms in Determination of Object-Oriented Programs Distribution on Homogenous Dedicated Networks

Saeed Parsa, Omid Bushehrian

Abstract

In this paper it is shown that the conventional cohesion and coupling criteria are not appropriate for the evaluation of the quality of the architecture of computationally intensive distributed code. This kind of architecture is aimed at achieving the shortest execution time by maximizing the degree of concurrency in execution of the distributed code. To achieve this, a new relation as a target function for estimating the overall execution time of computationally intensive object oriented code is presented. This relation is constructed while traversing the program call flow graph and considers both the sequential and asynchronous type for each call within the call flow graph. The type of the call is determined after the class dependency graph extracted from the distributed code is clustered. The aim is to find a clustering which maximizes concurrency in the execution of the intended program code.

Keywords

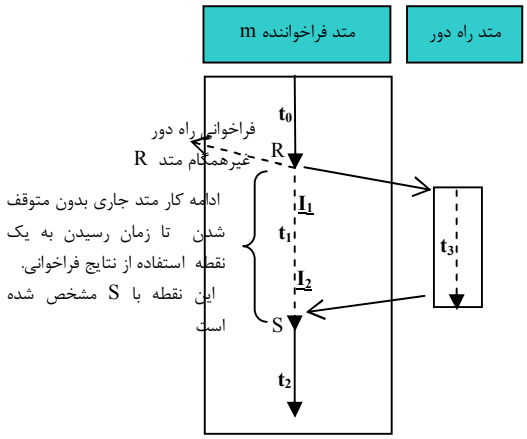
Distributed Computing, Clustering Algorithms, Time Estimation, Modularization

۱- مقدمه

هدف، افزایش سرعت اجرای برنامه های محاسباتی توزیع شده در سطح شبکه خاص منظوره* است. روش، بدست آوردن معماری مناسب بر اساس روابط بین کلاسهای موجود در داخل برنامه مورد نظر می باشد. اصولا به واسطه تعدد کلاسها در برنامه های بزرگ شئ گرا تعیین ساختار پیمانه ای مناسب بر اساس میزان ارتباط کلاسها کاری پیچیده است. لذا، روشهای استخراج اتوماتیک معماری مناسب از متن برنامه ها بر اساس میزان وابستگی بین کلاسها مطرح می باشد[۱][۲][۳]. معیار ارزیابی و بازسازی معماری در این روشها بر طبق اصول پیمانه سازی کد دو معیار حداکثر همبستگی داخلی هر جزء و حداقل وابستگی بین اجزا یا قطعات معماری بوده است. در این روشها پس از آنکه گراف ارتباطی کلاسها از متن برنامه شئ گرا استخراج می شود، بر اساس تعداد فراخوانیهای بین کلاسها لبه های مربوطه برچسب زده می شود و گراف وزن دار حاصل خوشه بندی می شود. به واسطه NP بودن مساله خوشه بندی روشهای خوشه بندی مختلفی بر اساس الگوریتمهای مکاشفه ای ازجمله روش ژنتیک[۱][۴] یا تپه نورد[۵] استفاده شده است. در این مقاله نشان داده شده که در یک برنامه محاسباتی که به واسطه افزایش سرعت در سطح شبکه خاص منظوره توزیع شده، تعداد فراخوانیها نمی تواند معیار صحیحی جهت وزن دهی و خوشه بندی گراف ارتباطی باشد. معیار می بایست میزان افزایش سرعت† حاصل از اجرای همروند متدهای وابسته در بین کلاسها باشد.

همواره تبدیل فراخوانیها از حالت ترتیبی به فراخوانی غیرهمگام راه دور نمی تواند موجب افزایش سرعت اجرایی برنامه ها باشد. همانگونه که در مرجع [۶] مشخص شده این تبدیل می تواند تاثیر منفی در سرعت اجرایی برنامه داشته باشد. در روش ارایه شده در این مرجع، ابتدا فرض شده که هر کلاس در گراف ارتباطی کلاسها در یک قطعه مجزا قرار دارد و سپس وزن یالهای بین هر دو کلاس را که معرف فراخوانیهای متد بین آنها می باشد، محاسبه می نماید. این محاسبه بدین صورت انجام می گیرد که هر فراخوانی متد بین دو کلاس مورد نظر اگر به صورت غیرهمگام‡ انجام پذیرد چه مقدار سبب افزایش سرعت کل برنامه می شود. این وزن می تواند عددی مثبت یا منفی باشد.

همانگونه که که در شکل (۱) مشخص شده است، میزان همروندی در اجرای فراخواننده و تابع فراخوانده شده وابسته به فاصله زمانی ای است که بین نقطه فراخوانی و نقطه استفاده از نتایج فراخوانی است. مشکل تخمین این فاصله زمانی است. همانگونه که در شکل ۱ مشخص شده،در فاصله بین نقطه فراخوانی و نقطه استفاده از نتایج فراخوانی ممکن است فراخوانیهای دیگری وجود داشته باشد که اجرای این فراخوانیها خود می تواند همگام یا غیرهمگام باشد.



شکل (۱): وجود فراخوانیهای متعدد در هنگام اجرای یک فراخوانی غیرهمگام.

۲- توزیع بهینه برنامه

چنانچه بیان شد، هدف، بدست آوردن توزیعی از کد برنامه است که حداکثر همروندی در اجرای اجزا توزیع شونده و درنتیجه حداکثر سرعت ممکن برای اجرای برنامه را حاصل نماید. به عبارت دیگر کلاسهای موجود در برنامه شیی گرا می بایست به نحوی در بین قطعات برنامه تقسیم شوند که اولاً میزان همروندی در اجرای متدهای کلاسها به حداکثر ممکن افزایش یابد و ثانیاً به منظور کاهش اثر تاخیر ارتباطات شبکه ای در زمان اجرای کل برنامه، تعداد فراخوانیهای راه دور به حداقل ممکن کاهش داده شود. همروندی در اجرای قطعات توزیع شده بواسطه فراخوانیهای غیرهمگام در بین قطعه های توزیع شده حاصل می گردد. به منظور تخمین میزان همروندی ایجاد شده در یک برنامه توسط یک توزیع، باید بتوان زمان اجرای برنامه را با توجه به چگونگی آن توزیع تخمین نمود. بدست آوردن زمان اجرایی کلیه دستورالعملها به غیر از فراخوانیهای تو در تو، توسط روشهای موجود قابل انجام است[۷][۸]. علت اینکه نمی توان روشهای موجود را برای محاسبه زمان اجرایی فراخوانی های تو در تو به سادگی در اینجا اعمال نمود این است که زمان اجرایی یک متد فراخوانی شده وابسته به اینکه فراخوانیهای داخل آن به صورت همگام یا غیر همگام انجام شوند متغیر است. به عنوان نمونه، شکل ۱ را در نظر بگیرید، در این شکل، زمان تخمینی برای متد فراخواننده m وابسته به اینکه متد R همگام یا غیرهمگام اجرا شود به صورت رابطه ۱ یا ۲ خواهد بود:

$$T_m = t_0 + T_R + t_1 + t_2 \,,\; T_R = t_3 \qquad (۱)$$

$$T_m = t_0 + t_1 + T_S + t_2 \,,\; T_S = \max(t_3 - t_1, 0) \qquad (۲)$$

دو حالت فوق را می توان در رابطه تخمین زمانی ۳ خلاصه نمود:

---

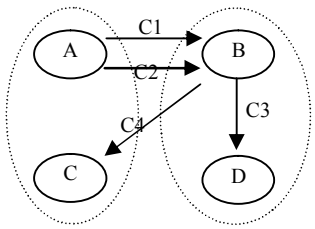
* Dedicated network
† speedup
‡ Asynchronous

$$T_m = t_0 + a_1 * T_R + t_1 + (1-a_1) * T_S + t_2 \ , \quad T_S = \max(t_3 - t_1, 0) \qquad (۳)$$

در رابطه فوق مقدار a_1 برحسب اینکه فراخوانی همگام یا غیرهمگام باشد 1 یا 0 در نظر گرفته می شود. با توجه به اینکه در یک متد تعداد زیادی فراخوانی متد ممکن است وجود داشته باشد، رابطه محاسبه زمانی می تواند بسیار پیچیده باشد. رابطه ریاضی تخمین زمان اجرایی برنامه ها با متدهای تو در تو در بخش بعدی ارایه شده است.

۳-رابطه کلی تخمین زمان اجرایی

می توان هر توزیعی از کد یک برنامه را بر مبنای میزان همروندی در کد برنامه که بواسطه فراخوانیهای راه دور غیر همگام حاصل می شود تخمین زد. مسلماً اگر در یک توزع برنامه دو کلاس از برنامه در دو قطعه مختلف توزیع شوند، فراخوانیهای بین آن دو کلاس تبدیل به فراخوانیهای راه دور می گردد. هدف بدست آوردن مجموعه ای از فراخوانیهای غیر همگام راه دور است که میزان همروندی در اجرای کد برنامه را به حداکثر ممکن و بالنتیجه زمان اجرای برنامه را به حداقل ممکن برساند. در ادامه به این مجموعه با عنوان *remote-call* ارجاع شده است. هر مجموعه *remote-call* شاخص یک توزیع از کلاسها در برنامه شیی گرای مورد نظر می باشد. هر توزیعی به واسطه یک خوشه بندی از گراف ارتباطی کلاسها که از کد برنامه شیی گرای توزیع شونده استخراج شده حاصل شده است. در واقع به واسطه هر خوشه بندی فراخوانیهایی که در داخل خوشه قرار می گیرند همگی به فراخوانیهای معمولی و فراخوانیهای بین خوشه ای تبدیل به فراخوانیهای راه دور غیر همگام می شوند. برای نمونه در شکل ۲ یک خوشه بندی و مجموعه *remote-call* حاصل از آن مشخص شده است. ارزش هر خوشه بندی بر اساس رابطه ارایه شده در بخش ۳-۱ تخمین می شود.



شکل ۲- معماری توزیعی نمونه. مجموعه remote-call متناظر $r=\{c_1,c_2,c_4\}$ می باشد.

۳-۱-گراف جریان فراخوانی

برای تخمین زمان اجرایی یک توزیع از برنامه داده شده، می بایست دنباله فراخوانیهای تو در تو را در برنامه مشخص نمود. برای این منظور گراف جریان فراخوانیها[§] مشخص می شود. در این گراف جهت دار گره ها شاخص متدها، لبه ها شاخص فراخوانی بین متدها و جهت لبه ها نمایانگر جهت فراخوانی است. در داخل هر گره فراخوانیها و نقاط استفاده از حاصل هر فراخوانی و زمان اجرایی سایر دستورالعملها به غیر از فراخوانی ها مشخص می شود. زمان اجرایی برای مجموعه دستورالعملهای به ترتیب از نقطه شروع متد مربوطه تا اولین فراخوانی زمان اجرایی بین هر فراخوانی و اولین نقطه استفاده از آن و بالاخره زمان اجرایی بین نقطه استفاده تا فراخوانی بعدی و یا انتهای متد به صورت جداگانه محاسبه و در گره مربوطه در داخل گراف جریان فراخوانی قرار داده می شود. مساله تعیین زمان اجرایی متدهای مورد فراخوانی است. قبل از پرداختن به این نکته باید خاطر نشان شد که در گراف جریان فراخوانی بواسطه فراخوانیهای خود بازگشتی ممکن است حلقه هایی وجود داشته باشد، باید حلقه ها را تشخیص داده تعداد تقریبی تکرار آنها را بدست آورد [۷] و سپس حلقه ها را از داخل گراف حذف نمود. بنا براین در داخل هر گره در گراف فراخوانیها می بایست تعداد تقریبی دفعاتی که کد متدهای مورد فراخوانی از طریق آن متد تکرار می شود را نیز مشخص کرد. گراف جریان فراخوانیها را می توان به صورت ذیل تعریف کرد:

$G = (E,V)$
$V =$ **seq** of element
element $\in \{I_i, S_i, W_i\}$

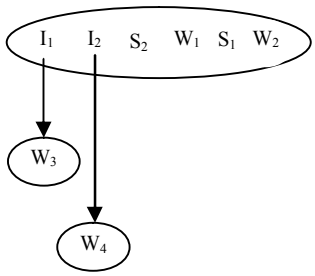
در تعریف فوق، E مجموعه لبه های گراف و V مجموعه گره های گراف می باشد. هر گره V شاخص بدنه یک متد می باشد و هر لبه E بین دو گره شاخص فراخوانی گره انتهایی لبه توسط گره ابتدای لبه جهت دار است. هر گره V یا در واقع هر متد در داخل گراف توسط دنباله ای مرتب از عناصر مشخص می شود. این عناصر می توانند یکی از سه نوع : جمله فراخوانی، جملات غیر فراخوانی و نقاط همزمانی که به ترتیب با I_i، W_i و S_i نشان داده شده اند، باشند. گره از نوع I_i حاوی نشانگر به گره دیگری از گراف می باشد که متد مربوط به آن از طریق متد کنونی مورد فراخوانی قرار گرفته است. گره از نوع S_i نمایانگر اولین محل استفاده از فراخوانی I_i می باشد. هر محل استفاده حاوی دو نشانگر می باشد. یکی از نشانگرها به جمله فراخوانی مربوطه اشاره می کند و نشانگر دیگر به عنصر بعدی در گره کنونی ارجاع می کند. گره از نوع W_i شاخص زمان تخمینی اجرای دنباله از جملات که حاوی جمله فراخوانی نیستند می باشد. برای نمونه در شکل ۳ یک برنامه نمونه و گراف جریان فراخوانی مربوطه مشخص شده است.

لازم به ذکر است که زمان تخمینی اجرا برای هر دستورالعمل در زبان جاوا بر مبنای واحد تعداد سیکل برای بایت کد معادل آن دستورالعمل، درنظر گرفته شده است[۸]. برای تخمین زمان اجرای حلقه ها از روشهای WCET [۷] استفاده نموده ایم. هر لبه در گراف یک برچسب فراخوانی I_i را به گرهی که معرف متد مقصد فراخوانی می باشد متصل می نماید. درصورتیکه I_i در داخل یک حلقه با تعداد تکرار

§ Call Flow Graph

n باشد، برچسب آن با مقدار n بالانویس می شود بدین معنی که این فراخوانی n مرتبه انجام می شود. در صورتیکه S_i در داخل حلقه ای با m تکرار باشد و این حلقه خود شامل I_i باشد برچسب S_i با مقدار m بالانویس می شود. در شکل ۳ یک برنامه نمونه و گراف جریان فراخوانی آن نشان داده شده است.

```
Calss A{                                        Calss B{
……  public static void main(string[]           …..
arg){                                           public int m(){
      int r1,r2;                                 //some
       B b=new B(); C c=new C();                 computations
       r1= b.m(); // I1                          }}//Class
       r2= c.n (); // I2                         ------------------
       use(r2);    //S2                          Calss C{
       use(r1);    //S1                          …..
      }                                           public int n(){
private void use(int r){…}                         //some
 }//Class                                         computations
                                                  }}//Class
```



شکل ۳- برنامه نمونه و گراف جریان فراخوانی متناظر

برای استخراج گراف جریان فراخوانی، متن برنامه باید با شروع از متد main آنالیز شود. در این آنالیز کلیه دستورات if نادیده گرفته می شوند.

```
Algorithm EFG (P: Program)
OUT:  DAG
Begin

EFG ← {}
Q ← {}
rootNode ← main() method of P
EFG.Connect(null, rootNode )
For each invocation I in rootNode.Elements()
    Q.add(I)
While Q is not empty
        I ← Q.getNext()
        node ← Target(I)
        EFG.Connect(I,node)
        For each invovation J in node.Elements()
                Q.add(J)
Wend
return EFG
End
```

شکل ۴-الگوریتم استخراج گراف جریان فراخوانی

الگوریتم استخراج گراف جریان فراخوانی در شکل ۴ آمده است. در این الگوریتم برای هر گره گراف یک خصوصیت بنام Elements() تعریف می شود که حاوی مجموعه عناصر آن گره (فراخوانی، محاسبه و همزمانی) می باشد.

۳-۲- رابطه تخمین زمانی

هدف بدست آوردن مجموعه ای از فراخوانیها درون یک برنامه است که اگر آن فراخوانیها به صورت غیر همگام راه دور انجام شوند زمان اجرایی به حداقل ممکن تقلیل یابد. در این راستا چنانچه مجموعه انتخاب شده r نامیده شود، رابطه ای مثل $EET_l(r)$** را معرفی می کنیم که برای هر فراخوانی *l* با در نظر گرفتن مجموعه r، زمان اجرایی را تخمین نماید. این رابطه، در حین پیمایش گراف جریان فراخوانی و با در نظر گرفتن اینکه هر فراخوانی ممکن است به صورت راه دورغیر همگام و یا به صورت همگام انجام شود، تشکیل داده می شود .

به عنوان مثال رابطه (۴) در ذیل شاخص رابطه *EET* برای گراف شکل ۳ می باشد:

$$EET_{main}(r)= a_1*EET_{I1}(r) + a_2*EET_{I2}(r) + (1-a_2)*T(S_2)+ w_1 + (1-a_1)*T(S_1)+ w_2$$
$$EET_{I1}(r)=w_3 , EET_{I2}(r)=w_4 , \qquad (۴)$$

در رابطه فوق ضرایب a_1 و a_2 با توجه به اینکه فراخوانیهای I_1 و I_2 همگام یا غیرهمگام باشند به ترتیب ۱ یا ۰ درنظر گرفته می شوند. S_1 و S_2 نقاط همزمانی یا به عبارت دیگر نقاط انتظار برای مقادیر برگشتی از فراخوانیهای I_1 و I_2 به ترتیب هستند. فرم کلی رابطه *EET* به صورت زیر می باشد:

$$EET_m(r)=\sum w_i + \sum a_i* EET_{Ii} (r) + \sum (1-a_i)*T(S_i) \qquad (۵)$$

در رابطه فوق ضرایب a_i برای فراخوانیهای I_i با نقاط انتظار S_i به صورت زیر مشخص می شوند:

$$a_i=\begin{cases} 1 & : I_i \notin r \\ 0 & : I_i \in r \end{cases}$$

در رابطه (۵)، $T(S_i)$ زمان تخمینی انتظار برای دریافت نتیجه فراخوانی غیرهمگام I_i است. البته این زمان تخمینی چنانچه فراخوانی I_i همگام و یالنتیجه $a_i=1$ باشد، صفر است. مقدار $T(S_i)$ با استفاده از رابطه ۶ محاسبه می شود:

$$T(S_i)= max((EET_{Ii} (r) + O_i) - t_i ,0) \qquad (۶)$$

در رابطه فوق O_i سربار زمانی حاصل از انجام فراخوانی راه دور و غیرهمگام I_i می باشد و t_i زمان تخمینی بین نقطه فراخوانی I_i و نقطه انتظار S_i است. با توجه به این نکته که حلقه ها در آغاز کار از گراف جریان فراخوانی حذف شده اند، با جایگزینی خود بازگشتی مقادیر *EET* نهایتا می توان به رابطه ای رسید که تنها حاوی ضرایب a_i، مقادیر تخمینی زمان W_i، سربار ارتباطی O_i و عملگرهای *max* باشد. در ویژگی ۱ این مساله نشان داده شده است. مساله یافتن توزیع بهینه

** Estimated Execution Time

را می توان معادل با مساله یافتن پیمانه سازی C از کلاسهای برنامه دانست که تابع کیفیت $EET_{main}(r)$ را که در آن *main* بیانگر متد آغازین برنامه و *r* بیانگر مجموعه فراخوانیهای غیرهمگام راه دور متناظر با C می باشد، کمینه نماید.

ویژگی ۱. صرف نظر از ترتیب نقاط فراخوانی و نقاط انتظار در یک گره n از گراف جریان فراخوانی که مقصد فراخوانی I است، رابطه $EET_I(r)$ قابل ساده شدن به رابطه ای است که فقط شامل W_i ها، ضرایب a_i ، مقادیر سربار O_i ومقدار *EET* برای فراخونیهای درون گره n باشد.

اثبات. براساس رابطه ۶ باید بتوانیم مقادیر T(S) را محاسبه کنیم. با توجه به اینکه ترتیب برچسبها در هر گره از چپ به راست قرارداد می شود، برای هر برچسب S، برچسب I متناظر آن در سمت چپ آن قرار دارد. برای محاسبه T(S) ها از سمت چپ ترین S، که S_i می نامیم، شروع می کنیم. مقدار $T(S_i)$ بر اساس رابطه ۶ ، وابسته به تخمین زمانی فاصله I_i تا S_i ، که همان t_i است، می باشد. با توجه به اینکه S_iچپ ترین نقطه همزمانی است، در این فاصله فقط برچسب های I و W وجود خواهند داشت. بنابراین محاسبه $T(S_i)$ بر اساس مقادیر W_i ها، ضرایب a_i ، مقادیر سربار O_i ومقدار EET برای فراخونیهای درون گره n ، قابل انجام است. برای دومین S از سمت چپ(S_j)، فاصله I_j تا S_j غیر از برچسب های I و W، شامل یک برچسب S_i می باشد که قبلا مقدار $T(S_i)$ برای آن محاسبه شده است. به همین ترتیب مقادیر T(S) برای سایر S ها نیز قابل محاسبه خواهد بود.

برای نشان دادن این مساله که چگونه یک رابطه *EET* برای ارزیابی میزان مناسب بودن یک توزیع بکار می رود برنامه نمونه شکل ۳ را درنظر بگیرید. رابطه *EET* برای این برنامه به صورت زیر محاسبه می شود:

$$EET_{main}(r)= a_1*EET_{I1}(r) + a_2*EET_{I2}(r) + (1-a_2)*T(S_2) + w_{use} + (1-a_1)*T(S_1) + w_{use}$$

$$EET_{I1}(r)=w_m ,\ EET_{I2}(r)=w_n ,$$

$$T(S_2)= \max((EET_{I2}(r) + O_2) - t_2 ,0) = \max(w_n+O_2-t_2,0) ,$$

$$t_2=0 \Rightarrow T(S_2)=w_n + O_2$$

$$T(S_1)= \max((EET_{I1}(r) + O_1) - t_1 ,0) = \max(w_m+O_1-t_1,0) ,$$

$$t_1=w_{use} + (1-a_2)*T(S_2) + a_2*EET_{I2}(r)= w_{use} + (1-a_2)*(w_n + O_2) + a_2*w_n$$

با جایگزین نمودن جملات *EET* نهایتا به رابط ای می رسیم که تنها شامل w_m, w_n, w_{use} ، سربار ارتباطی O_1, O_2 و ضرایب a_i می باشد. به توسط این رابطه می توان مقدار بهینه یک خوشه بندی را محاسبه نمود. برای برنامه نمونه شکل ۳ یک توزیع می تواند بدین صورت باشد که کلاسهای *A* و *B* در یک خوشه باشند حال آنکه C در خوشه جداگانه ای باشد. برای این خوشه بندی مجموعه *remote-call* به صورت $\{I_2\}$ خواهد بود. بنابراین خواهیم داشت $a_1=1, a_2=0$:

$$EET_{main}(r_1)=w_m + w_n + O_2 + w_{use} + w_{use} ,$$

Let: $w_m=w_n=w_{use}=w$ and $O_1=O_2 =O \Rightarrow$
$EET_{main}(r_1)= 4w + O$

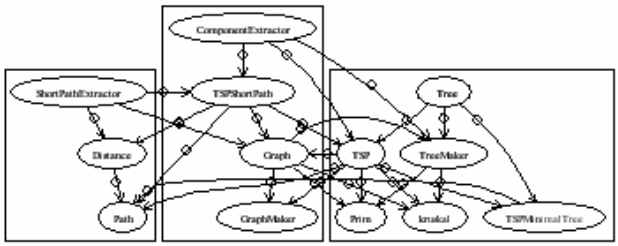
حال، خوشه بندی دیگری که که در آن A و C در یک خوشه هستند و B در خوشه متفاوت در نظر بگیرید: a1=0, a2=1

$$EET_{main}(r_2)=w_n + w_{use} + \max(w_m+O_1-w_{use}-w_n ,0) + w_{use} = 3w + \max(O-w ,0)$$

با توجه به اینکه $EET_{main}(r_2) < EET_{main}(r_1)$ خوشه بندی دوم انتخاب می شود.

۴-مورد مطالعه

در این بخش توزیع بهینه برنامه TSP که به زبان جاوا نوشته شده است را توسط رابطه زمانی بیان شده در این مقاله استخراج می نماییم. این برنامه که شامل هجده کلاس است، برای استخراج تور هامیلتونی نزدیک به بهینه در یک گراف، توسط درخت پوشای مینیمم آن گراف، نوشته شده است. کلاس TSP ، شامل فراخوانی به متد از کلاسهای prime و kruskal می باشد که قسمت عمده زمان اجرای برنامه را تشکیل می دهند. در این دو کلاس عملیات ساختن درختهای پوشای مینیمم به دو روش پیاده سازی شده است. با توجه با اینکه فراخوانی به متدهای این دو کلاس درون حلقه قرار دارد، استفاده از معیار ماگزیمم چسبندگی و مینیمم اتصال در آن سبب قرار گرفتن کلاسهای TSP، Prime و kruskal در یک خوشه می شود(شکل ۵).



شکل ۵-خوشه بندی با معیار ماگزیمم چسبندگی و مینیمم اتصال

رابطه تخمین زمانی برای قسمتی از متد main از کلاس TSP که مربوط به فراخوانی به کلاسهای prime و kruskal می باشد به صورت زیر می باشد:

$$EET_{main}(r) = a_1*I_1 + a_2*I_2 + (1-a_2)*T(S_2) + a_3*I_3 + (1-a_3)*T(S_3) + (1-a_1)*T(S_1)$$

که در رابطه فوق فراخوانیهای I_1، I_2 و I_3 به ترتیب معرف فراخوانی به متد کلاس prime، kruskal و path می باشند. زمان اجرایی فراخوانیهال I_1 ، I_2 و I_3 بر حسب واحد ثانیه به ترتیب ۱۶۰ ، ۷۰ و ۱ می باشند. با در نظر گرفتن سربار ارتباطی در فراخوانی راه دور به میزان ۱ ثانیه برای مقادیر مختلف a_i مقدار محاسبه شده برای $EET_{main}(r)$ را به صورت جدول ۱ خواهیم داشت:

جدول ۱- مقایسه مجموعه های مختلف remote-call

مجموعه r	مقدارتخمینی زمان اجرا (ثانیه)
$a_1=0, a_2=0, a_3=0$	161
$a_1=0, a_2=0, a_3=1$	161
$a_1=0, a_2=1, a_3=0$	161
$a_1=0, a_2=1, a_3=1$	161
$a_1=1, a_2=0, a_3=0$	233
$a_1=1, a_2=0, a_3=1$	232
$a_1=1, a_2=1, a_3=0$	232
$a_1=1, a_2=1, a_3=1$	231

با توجه به جدول فوق، کوتاهترین زمان اجرا هنگامی بدست می آید که متد I_1 غیرهمگام اجرا شود(سطرهای اول تا چهارم). با توجه به شرط کمترین تعداد فراخوانی غیرهمگام سطر چهارم به عنوان توزیع بهینه انتخاب می شود. در این حالت فراخوانی I_1 به صورت غیرهمگام انجام می شود درحالیکه فراخوانیهای I_2 و I_3 به صورت همگام انجام می گردد. درنتیجه کلاسهای TSP و Prime باید در خوشه های مختلف قرار بگیرند. که با خوشه بندی ارایه شده در شکل ۵ که با معیار ماگزیمم چسبندگی و مینیمم اتصال بدست آمده است متفاوت است.

۴-نتیجه گیری

مساله اصلی در بدست آوردن توزیعی از برنامه شیی گرا که حداقل زمان اجرای برنامه را حاصل نماید، بدست آوردن یک رابطه تخمین زمانی کلی برای برنامه است. با استفاده از این رابطه باید بتوان زمان اجرای برنامه را با در نظر گرفتن اینکه فراخوانیها ممکن است به صورت غیرهمگام و یا ترتیبی انجام شوند و همچنین زمان انتظار برای دریافت نتایج فراخوانیهای غیر همگام، تخمین زد. در این مقاله چنین رابطه ای از روی گراف جریان فراخوانی برنامه بطور اتوماتیک تولید می شود. نکته مهم در تولید این رابطه تخمین زمانی، این است که نوع فراخوانیها قبل از خوشه بندی گراف فراخوانی مشخص نیست. لذا در هنگام ایجاد رابطه تخمین زمان اجرایی فراخوانیهای تو در تو، با استفاده از یک پارامتر بولین برای هر فراخوانی هر دو حالت راه دورغیر همگام و ترتیبی برای فراخوانی در نظر گرفته می شود. مقدار این پارامترهای بولین پس از خوشه بندی گراف جریان فراخوانی مشخص می شود. مساله بدست آوردن خوشه بندی ای است که حداکثر همروندی در اجرای برنامه توزیعی را میسر نماید.

مراجع

[1] Parsa S, Bushehrian O.: The Design and Implementation of a Tool for Automatic Software Modularization, Journal of Supercomputing, Volume 32, Issue 1, April 2005.
[2] S. Parsa, O. Bousherian, ”A Framework to Investigate and Evaluate Genetic Clustering Algorithms for Software modularization”, LECTURE NOTES IN COMPUTER SCIENCE, Vol. 3037 (June 2004), SPRINGER-VERLAG.
[3] S. Mitchell Brian, Mancoridis Spiros, ”Bunch: A Clustering tool for the Recovery and Maintenance of Software System Structure”, in Proc. of International Conf. of Software Maintenance, IEEE 1999.
[4] S. Mitchell Brian, “A Heuristic Search Approach to Solving the Software Clustering Problem”, Thesis, Drexel University, March 2002.
[5] S. Mitchell Brian,Mancoridis Spiros, ”CRAFT: A Framework for Evaluating Software Clustering Results in the Absence of Benchmark Decomposition”, in Proc. of IWPC, IEEE 2001.
[6] Saeed Parsa, Vahid Khalilpour, “Automatic Distribution of Sequential Code Using JavaSymphony Middleware”, SOFSEM06, Lecture Notes in Computer Science, 2006
[7] Martin Schoeberl, Rasmus Pedersen, “WCET Analysis for a Java Processor“, Java Technologies for Real-time and Embedded Systems 2006 Paris France ACM 2006.
[8] Martin Schoeberl, “Using a Java Optimized Processor in a Real World Application”, WISES 2003, pp165-176