طراحی و ساخت یک میان‌افزار برای سیستم‌های توزیع‌شده تحت .NET Framework

به عنوان

پروژه پایان ترم درس سیستم های توزیع شده

جهت ارائه به

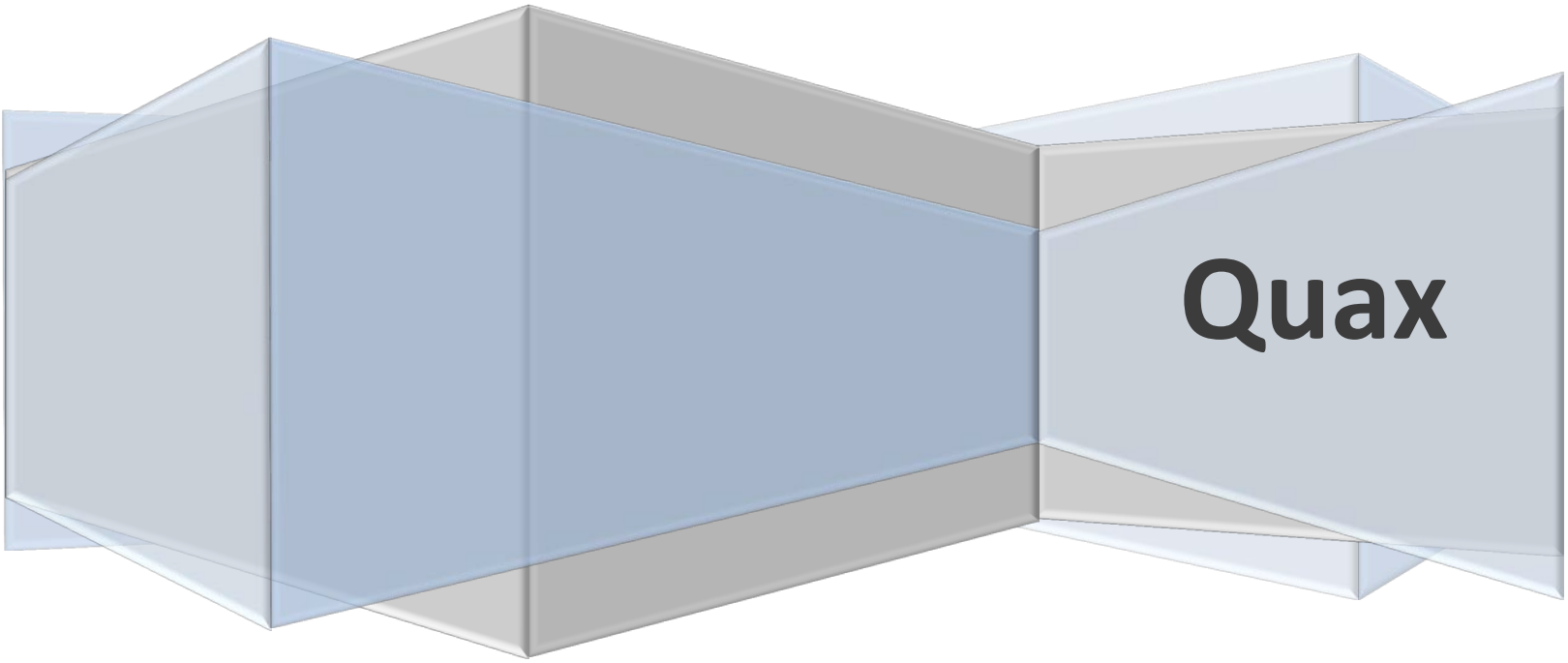
سرکار خانم دکتر لیلی محمدخانلی

احمد تاجدینی

کاوه دولت‌آبادی

سیامک عبداله‌زاده

دانشگاه تبریز، گروه علوم کامپیوتر، بهار 1391



1. مقدمه

هدف از ساخت یک میان‌افزار[1] در سیستم‌های توزیع شده این است که انتزاعی ایجاد کنیم که مجموعه‌ای از کامپیوترهای مستقل از هم به صورت یک کامپیوتر یکپارچه و قدرتمند دیده شوند. یک میان‌افزار با ایجاد انواع مختلفی از شفافیت[2] این امکان را به برنامه کاربردی می‌دهد تا بدون درگیر شدن با جزئیات، از منابع تحت مدیریت میان افزار استفاده کند (منابعی مانند CPU، حافظه اصلی و جانبی و غیره).

میان افزارهای تجاری مختلفی در بازار موجود است. برخی از آن‌ها کاملا حرفه‌ای و دارای رابط برنامه نویسی برنامه کاربردی[3] پیچیده‌ای می‌باشند. هدف ما از ساخت این نرم‌افزار این بود که اولا میان‌افزاری ایجاد کنیم که در عین سهولت استفاده و سادگی، قابلیت انعطاف زیادی نیز داشته باشد تا بتواند سناریوهای مختلف محاسباتی را پشتیبانی کند. ثانیا تحت معماری .NET Framework باشد، چون با بررسی های که انجام دادیم به این نتیجه رسیدیم که در حال حاضر میان‌افزار مناسبی برای سیستم‌های توزیع شده تحت این چارچوب وجود ندارد و نرم‌افزارهای موجود نیز یا دارای مستندات کاملی نیستند و یا دیگر توسط گروه و یا شرکت سازنده آن به روزرسانی و پشتیبانی نمی‌شوند.

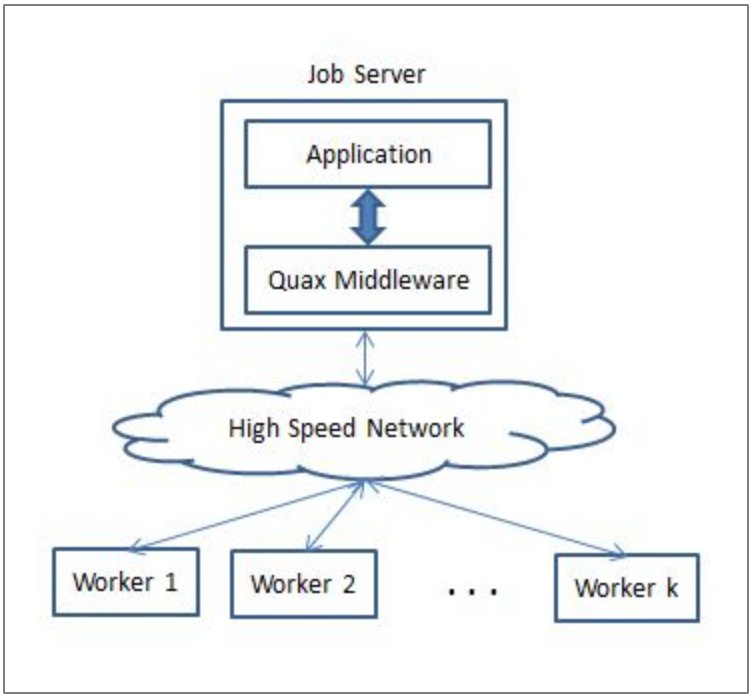
میان‌افزاری که ما طراحی کرده و ساخته‌ایم، اساسا دو نوع شفافیت را ایجاد می کند. مورد اول، شفافیت در دسترسی به منابع کامپیوترهای دیگر است (Access Transparency). مورد دوم شفافیت در از کار افتادگی کامپیوترهایی است که تحت نظر میان‌افزار کار می‌کنند (Failure Transparency). اکثر رویدادهایی که در سطوح داخلی میان‌افزار یا در سطح شبکه رخ می‌دهد، به صورت گزارش (Log) در اختیار نرم افزار کاربردی که از میان‌افزار استفاده می‌کند قرار داده می شود، البته هیچ الزامی وجود ندارد که نرم‌افزار کاربردی از این اطلاعات استفاده کند.

به منظور اینکه در ادامه به آسانی به نرم‌افزاری که ساخته ایم ارجاع داشته باشیم، آن را **"Quax"** نام‌گذاری کرده‌ایم.

1. Middleware 2. Transparency 3. API

2. معماری کلی میان‌افزار Quax

معماری کلی میان‌افزار Quax مبتنی بر سیستم صف است. فرض بر این است که برنامه کاربردی دارای پردازش‌های نسبتا سنگین است که مایل است آن‌ها را روی چندین کامپیوتر توزیع کند و آن‌ها را به صورت همزمان اجرا کند و پس از به پایان رسیدن پردازش ها، نتایج آن‌ها را دریافت کند. میان‌افزار Quax سناریو ذکر شده را پشتیبانی می کند. در شکل 1 نمای کلی این سیستم محاسباتی توزیع شده را ملاحظه می‌فرمایید.

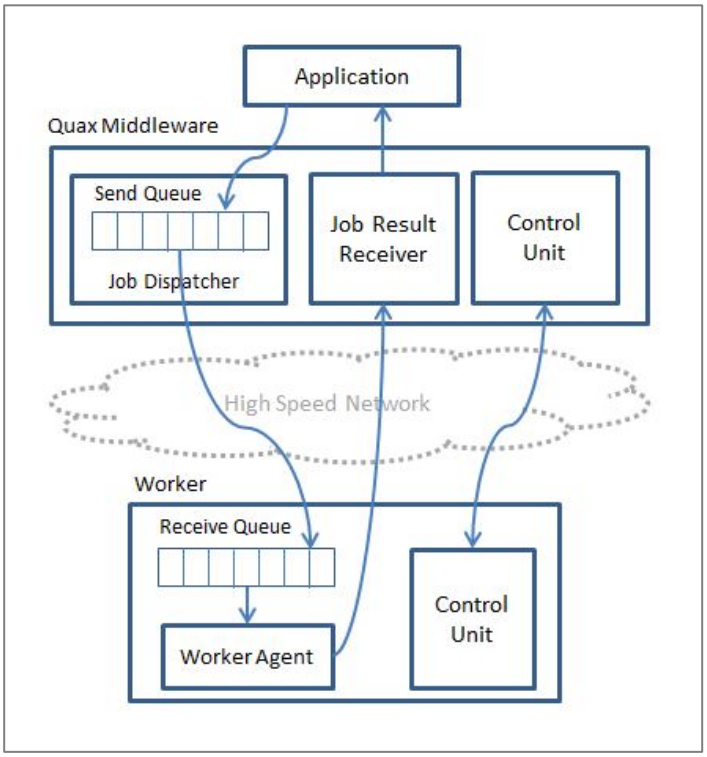


شکل 1. نمای کلی یک سیستم محاسباتی توزیع شده برمبنای میان‌افزار Quax

به هرکدام از پردازش هایی که برنامه کاربردی مایل است آن‌ها را توزیع کند یک Job می‌گوییم. یک Job شامل یک فایل اجرایی (قابل اجرا توسط سیستم عامل) و یک فایل ضمیمه است که شامل داده‌های ورودی مورد نیاز برای اجرای Job است. مجموعه برنامه کاربردی و میان‌افزار روی کامپیوتری قرار می‌گیرند که به آن Job Server می‌گوییم. برنامه کاربردی یک یا چندین Job را به میان‌افزار تحویل می‌دهد، سپس میان‌افزار به یکی از کامپیوترهای (از پیش معرفی شده) تحت نظر میان‌افزار متصل می‌شود و یک یا چندین Job را به آن ارسال می‌کند. به این کامپیوترهای خاص Worker یا کامپیوتر کارگر می‌گوییم. وظیفه Worker این است که یک یا چندین Job را از میان‌افزار تحویل بگیرد و آن‌ها را اجرا کند و پس از به پایان رسیدن اجرا، نتایج حاصل از اجرای آن‌ها را به میان‌افزار ارسال کند. هر

Worker یک شناسه منحصر به فرد دارد. میان‌افزار Quax تا 65535 کامپیوتر کارگر را پشتیبانی می‌کند. ممکن است در حین اجرای یک Job روی یک Worker خاص، به طور ناگهانی Worker از کار بیافتد. در این حالت میان‌افزار باید دارای مکانیزم‌هایی باشد تا از این اتفاق مطلع شود و Jobهایی که به آن Worker ارسال کرده را مجددا به یک Worker دیگر ارسال کند (البته همین اتفاق نیز ممکن است برای Worker دوم رخ دهد).

برای اینکه کمی دقیق تر معماری میان‌افزار Quax را بررسی کنیم، شکل 2 را ملاحظه بفرمایید. همانطور که از این شکل مشاهده می شود این میان‌افزار شامل سه مولفه می‌باشد، *واحد گسیل کننده Job*[1]، *واحد دریافت کننده نتایج*[2] و *واحد کنترل*[3]. اصلی‌ترین مولفه این نرم‌افزار واحد گسیل کننده Job است. وظیفه این واحد این است که یک یا چندین Job را به یکی از Workerهای موجود ارسال کند (اینکه طبق چه الگوریتمی از بین Workerهای موجود یکی را انتخاب کنیم در ادامه بحث خواهد شد). همچنین گسیل کننده Job باید اطلاعات مربوط به اینکه کدام Job به کدام Worker ارسال شده است را به خاطر بسپرد تا اگر Worker ای خراب شد، بتواند Jobهای ارسال شده به آن را شناسایی کند و آن‌ها را مجددا به یک Worker دیگر ارسال کند.



شکل 2. معماری کلی میان‌افزار Quax

گسیل کننده Job دارای یک صف به نام صف ارسال[4] است. وقتی برنامه کاربردی یک Job جدید را به میان‌افزار تحویل می‌دهد، میان-افزار آن را در صف ارسال قرار می‌دهد. گسیل کننده Job در اولین فرصت یک یا چندین Job را از این صف خارج می‌کند و آن‌ها را به یک Worker ارسال می‌کند. برنامه کاربردی تا هر زمان و به هر میزان می‌تواند Job جدید تولید کند و آن‌ها را به میان‌افزار تحویل دهد.

1. Job Dispatcher 2. Job Result Receiver 3. Control Unit 4. Send Queue

گسیل کننده Job به صورت تدریجی Jobها را از صف خارج می‌کند و به Worker ها ارسال می‌کند. این فرایند تا زمانی که صف ارسال خالی نشده باشد ادامه می‌یابد.

مولفه بعدی این میان‌افزار دریافت کننده نتایج نام دارد. وظیفه این مولفه دریافت نتایج از Worker ها و تحویل آن‌ها به برنامه کاربردی می‌باشد. همچنین این مولفه باید بعد از دریافت نتایج یک Job، گسیل کننده Job را از این رویداد باخبر کند تا اطلاعات مربوط به آن Job را از حافظه خود پاک کند (گسیل کننده Job تا زمان دریافت نتایج اجرای یک Job اطلاعات آن Job را ذخیره می کند، زیرا ممکن است نیاز باشد که آن Job را به یک Worker دیگر ارسال کند). در بعضی مواقع ممکن است شرایطی پیش آید که به ازای یک Job چندین نتیجه یکسان از Workerهای مختلف دریافت شود(در ادامه به این موضوع خواهیم پرداخت). در این حالت دریافت کننده نتایج می‌بایست فقط یکی از جواب‌ها را در اختیار برنامه کاربردی قرار دهد و بقیه جواب‌ها را دور بریزد.

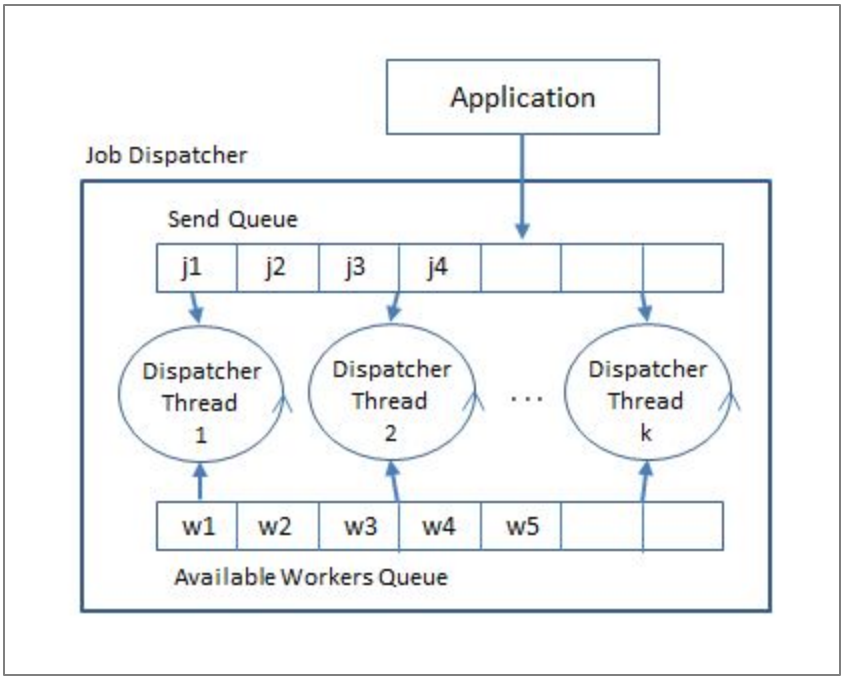
سومین مولفه میان‌افزار Quax واحد کنترل است. یکی از وظایف این مولفه ثبت[1] Worker های جدید و اضافه کردن آن‌ها به Workerهای موجود است. وظیفه دیگر این واحد بررسی وضعیت Workerهای موجود است. چنانچه بعد از سپری شدن یک مدت زمان خاص هیچ پاسخی از یک Worker دریافت نشود، این واحد با ارسال یک پیام کنترلی خاص به Worker از آن می خواهد تا "زنده بودن" خود را اعلام کند. چنانچه پاسخی دریافت نشود واحد کنترل فرض را بر این می‌گذارد که Worker از کار افتاده است و این موضوع را به اطلاع گسیل کننده Job می‌رساند تا در آینده Job دیگری به این Worker ارسال نشود. همچنین گسیل کننده Job باید Jobهای ارسال شده به این Worker را مجددا به صف ارسال اضافه کند تا در ادامه کار به یک Worker دیگر ارسال شوند.

3. تشریح عملکرد واحد گسیل کننده Job

همانطور که در بخش قبلی گفته شد وظیفه گسیل کننده Job ارسال Jobها به Workerها است. در اینجا عملکرد آن را دقیق‌تر بررسی می‌کنیم.

شکل 3 نمای کلی واحد گسیل کننده Job را نمایش می‌دهد. این واحد دارای یک صف ارسال، صف Available Workers و تعدادی نخ[3] می‌باشد. فرض کنید از قبل برنامه کاربردی تعدادی Job را به صف ارسال اضافه کرده است و همچنین تعدادی Worker نیز خودشان را به میان‌افزار معرفی کرده‌اند (و مشخصات آن‌ها ثبت شده است). این Worker ها را در صفی به نام Available Workers قرار می‌دهیم. این Worker ها آماده دریافت Job و اجرای آن می‌باشند.

1. Register 2. Job Result Receiver 3. Thread



شکل 3. نمای کلی واحد Job Dispatcher

برای ارسال Jobها یک روش ساده این است که یک نخ که به آن نخ گسیل کننده[1] می‌گوییم بسازیم. وظیفه این نخ این است که یک Job را از صف ارسال خارج کند، برای مثال j، و همچنین یک Worker را از صف Available Workers خارج کند، برای مثال w، و سپس j را به w ارسال کند. بعد از پایان ارسال w به انتهای صف Available Workers اضافه می‌شود و این فرایند را تا زمانی تکرار می‌کنیم که صف ارسال خالی شود. این روش دو نقطه ضعف دارد:

1. ضعف اول این روش این است که نخ گسیل کننده Job، هر بار که به یک Worker متصل می‌شود فقط یک Job را به آن ارسال می‌کند. فرض کنید صف ارسال حاوی یک هزار job باشد. و صف Available Workers فقط شامل ده Worker باشد. با روشی که شرح داده شد، نخ گسیل کننده باید به هر کامپیوتر کارگر $\frac{1000}{10} = 100$ بار متصل شود و در هر بار اتصال یک Job را به آن ارسال کند. بدیهی است که سربار ناشی از ایجاد مکرر اتصال به Worker مذکور باعث کاهش کارآیی می-شود. برای حل این مشکل میان‌افزار Quax در هر بار اتصال به یک Worker چندین Job را به صورت گروهی ارسال می‌کند. تعداد Jobهای ارسالی از رابطه زیر بدست می‌آید:

$$\text{Jobs Per Worker} = \frac{|\text{Send Queue}|}{|Available\ Workers|}$$

1. Dispatcher Thread

یعنی تعداد Jobهای موجود در صف ارسال تقسیم بر تعداد Workerها. بنابراین در مثال قبل با یک اتصال به هر Worker، 100 عدد Job را به صورت گروهی به آن ارسال می‌کنیم. البته ممکن است بعد از محاسبه این رابطه، تعدادی Job جدید به صف ارسال اضافه شود و یا اینکه تعدادی Worker جدید به صف Available Workers اضافه شود، در این صورت می‌بایست نسبت Jobs Per Worker را مجددا محاسبه کنیم.

2. مشکل دوم این روش این است که فقط یک نخ گسیل کننده دارد. در این حالت در هر لحظه مشغول ارسال اطلاعات به فقط یک Worker خواهیم بود و بقیه Workerها باید منتظر بمانند تا نوبت آن‌ها فرا برسد. این مشکل وقتی بیشتر پدیدار می‌شود که در اتصال به یک Worker مشکلی وجود داشته باشد. در این حالت ممکن است نخ گسیل کننده تمایل داشته باشد چندین بار برای برقراری اتصال مجددا تلاش کند. این موضوع باعث می شود که زمان Worker های دیگر (به دلیل انتظار) به هدر رود. روش بهتر این است که چندین نخ گسیل کننده داشته باشیم که همزمان با هم کار کنند. هر نخ گسیل کننده مجموعه‌ای از Jobها را از صف ارسال خارج می‌کند و آن‌ها را به یک Worker ارسال می‌کند. میان‌افزار Quax از چندین نخ گسیل کننده به صورت همزمان استفاده می کند. تعداد نخ‌ها را معمولا باید یک عدد کوچکتر از 20 یا 30 در نظر گرفت (زیرا منابع سیستم عامل محدود است و بهینه نیست که تعداد زیادی نخ ایجاد کنیم). همچنین تعداد نخ ها باید از تعداد Worker ها کمتر باشد، زیرا نخ‌های اضافی کمکی به افزایش کارآیی فرایند ارسال Job نمی‌کنند. تعداد نخ های گسیل کننده در میان‌افزار Quax به صورت پویا در زمان اجرا تعیین و در صورت لزوم کم یا زیاد می‌شود.

   از آنجایی که چندین نخ به منابع مشترک دسترسی دارند (صف ارسال و صف Available Workers) باید ملاحظات مربوط به همزمانی نخ[1] را نیز در نظر داشت. ما از Monitor برای همزمانی نخ ها استفاده کرده ایم.

بعد از ارسال یک یا چندین Job به یک Worker، گسیل کننده Job اطلاعات مربوط به این ارسال را در جدولی به نام (Pending Jobs) ذخیره می کند. در این جدول مشخص می کنیم که به هر Worker چه Jobهایی ارسال شده است. در شکل 4 ساختار این جدول را ملاحظه می فرمایید.

Worker ID	Jobs	Start Time	Expected Time To Complete	Max Time To Complete
1	1, 2, 3	12:23:35	30 seconds	1 minute
2	4, 5	12:23:37	3 minutes	9 minutes

شکل 4. جدول Pending Jobs (با 2 سطر برای نمونه)

1. Thread Synchronization

در ستون اول این جدول شناسه Worker ای قرار می‌گیرد که به آن Job ارسال کرده‌ایم. در ستون دوم شناسه Jobهای ارسال شده به آن Worker درج می‌شود (به مرور زمان ممکن است به این مجموعه شناسه Jobهای دیگری نیز افزوده شود). در ستون سوم زمان ارسال Jobها درج می‌شود. در ستون چهارم زمان مورد انتظار برای دریافت نتایج (Expected Time To Complete) مربوط به همه Jobهایی که ارسال شده است درج می‌شود. در واقع عددی که در این ستون درج می‌شود مشخص می‌کند که ما انتظار داریم در طول این مدت (برای مثال 30 ثانیه) نتایج همه Jobها را دریافت کنیم. در ادامه خواهیم گفت که چطور این عدد را محاسبه می‌کنیم. چنانچه بعد از سپری شدن این مدت پاسخی از طرف Worker مربوطه دریافت نکردیم از واحد کنترل می‌خواهیم که "زنده" بودن یا نبودن آن Worker را بررسی کند. اگر واحد کنترل تشخیص دهد که Worker مورد نظر خراب است، تمامی Jobهای اختصاص داده شده به این Worker را به صف ارسال برمی‌گردانیم تا بعدا به یک Worker دیگر ارسال شود. اگر "زمان مورد انتظار برای دریافت نتایج" سپری شد و واحد کنترل نیز تشخیص داد که Worker مربوطه "زنده" است، احتمالا یا تخمین ما از "زمان مورد انتظار برای دریافت نتایج" اشتباه بوده یا بار کاری Worker مورد نظر زیاد بوده و نتوانسته در طول این زمان پاسخ‌ها را ارسال کند. در این حالت باید به آن وقت بیشتری بدهیم تا پاسخ‌ها را ارسال کند. حداکثر مدت زمانی که برای دریافت جواب از یک Worker منتظر می‌مانیم را در ستون پنجم قید می‌کنیم (Max Time To Complete). اگر در طول این زمان جوابی از Worker دریافت نشود، فرض را بر این می‌گذاریم که Worker خراب شده است. در این حالت Jobهای ارسال شده به آن را مجددا به صف ارسال اضافه می‌کنیم تا در ادامه فرآیند ارسال، به یک Worker دیگر ارسال شوند. همچنین این Worker را به صف Available Workers برنمی‌گردانیم تا در آینده Job دیگری به آن ارسال نشود (تا زمانی که از آن Worker مجددا سیگنال اعلام آمادگی دریافت شود).

برای اینکه به ازای هر Worker زمان‌های "Expected Time To Complete" و "Max Time To Complete" را محاسبه کنیم لازم است به هر Job یک "مدت زمان مورد انتظار برای اجرا" و یک "حداکثر مدت زمان مورد نیاز برای اجرا" اختصاص دهیم. از آنجایی که برنامه کاربردی Jobها را تولید می‌کند، لذا احتمالا می‌تواند این زمان‌ها را تخمین بزند. اگر برنامه کاربردی نمی‌تواند تخمین نسبتا درستی از این زمان‌ها بزند، می‌تواند تخمین‌هایی با مقدار زمانی بالا بزند، یعنی زمان اجرای بدترین حالت را در نظر بگیرد).

برای اینکه "مدت زمان اجرای مورد انتظار" را برای یک Worker حساب کنیم می‌توانیم "مدت زمان اجرای مورد انتظار" همه Jobهایی که به آن ارسال کرده‌ایم را با هم جمع کنیم:

$$Expected\ Time\ To\ Complete\ (for\ worker\ w) = \sum_{\substack{for\ all\ jobs\ j\ sent \\ to\ worker\ w}} Expected\ execution\ time\ for\ job\ j$$

به همین نحو برای ستون "حداکثر مدت زمان اجرا" عمل می‌کنیم. به هر job یک "حداکثر مدت زمان اجرا" اختصاص می‌دهیم و مقدار ستون "Max Time To Complete" را از رابطه زیر بدست می‌آوریم:

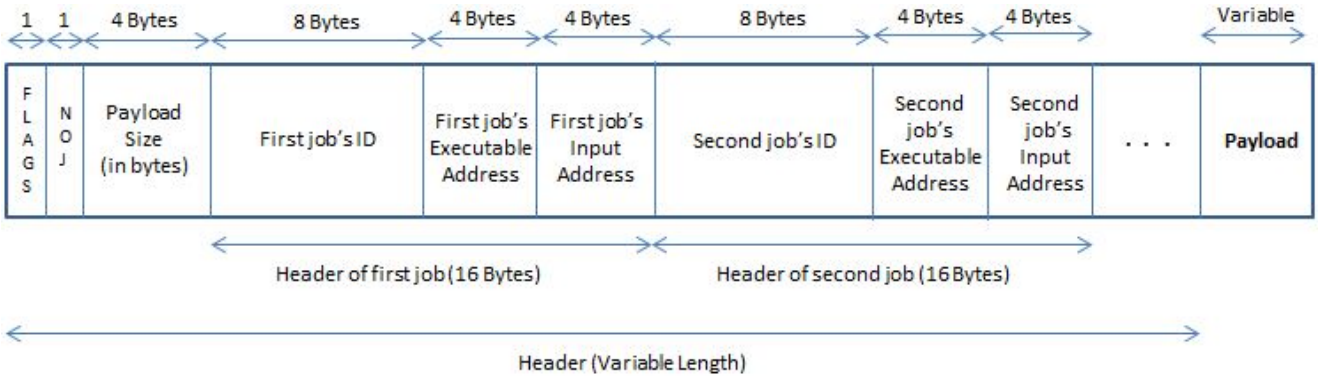
$$Max\ Time\ To\ Complete\ (for\ worker\ w) = \sum_{\substack{for\ all\ jobs\ j\ sent \\ to\ worker\ w}} Maximum\ execution\ time\ for\ job\ j$$

اگر واحد دریافت کننده نتایج رسیدن نتایج یک Job را گزارش کند، مقدار "زمان مورد انتظار برای اجرا" مربوط به آن Job از ستون "Expected Time To Complete" مربوط به Worker محاسبه کننده آن Job کم می شود.

اگر بعد از ثبت این اطلاعات در جدول Pending jobs، یک یا چند Job جدید تولید شود و به یکی از Workerهایی ارسال شود که شناسه آن در جدول Pending Jobs وجود دارد (یعنی قبلا به آن Jobای ارسال شده باشد و این Worker در حال حاضر مشغول اجرای آن باشد)، مقادیر ستون های "Expected Time To Complete" و "Max Time To Complete" مربوط به آن Worker با توجه به مقادیر "زمان مورد انتظار برای اجرا" و "حداکثر مدت زمان اجرا" مربوط به job جدید به روز رسانی می‌شود.

3.1. ساختار اطلاعات ارسال شده توسط گسیل کننده Job

بعد از اینکه نخ گسیل کننده تعدادی Job را از صف ارسال و یک Worker را از صف Available Workers خارج کرد، باید Jobها را به آن Worker ارسال کند. ابتدا گسیل کننده Job یک اتصال TCP با Worker مورد نظر برقرار می‌کند، سپس کل Jobها را به صورت یک بسته اطلاعاتی درآورده و ارسال می‌کند و در نهایت اتصال را قطع می‌کند. ساختار بسته ای که گسیل کننده Job ارسال می‌کند را در شکل 5 ملاحظه می‌فرمایید.



شکل 5. ساختار بسته ارسال شده توسط Job Dispatcher (گسیل کننده Job)

ساختار این بسته به شرح زیر است:

فیلد *Flags* رزرو شده است.

فیلد *NOJ* (**N**umber **O**f **J**obs): تعداد Jobهایی که در این بسته قرار گرفته‌اند را مشخص می‌کند.

فیلد *Payload Size*: سایز Payload را مشخص می‌کند (بر حسب بایت).

اولین 16 بایت بعد از فیلد Payload Size شامل اطلاعاتی در مورد اولین Jobای است که در این بسته قرار دارد. 16 بایت بعدی شامل اطلاعاتی در مورد Job دوم است و همینطور الی آخر. این ساختار 16 بایتی شامل 8 بایت مربوط به شناسه Job، 4 بایت مربوط به آدرس شروع فایل اجرایی Job و 4 بایت مربوط به آدرس شروع فایل ورودی می‌شود. محتویات فایل اجرایی و فایل ورودی مربوط به هر Job در قسمت Payload و پشت سر هم قرار می‌گیرند. آدرس‌هایی که در این فیلدها ذکر می‌شوند آدرس‌هایی در فضای آدرس Payload است و نه در فضای آدرس کل پیام.

تمامی اعدادی که در فیلدهای این بسته قرار می‌گیرد به صورت Little Endian ذخیره می‌شوند (بایت کم ارزش‌تر در آدرس پایین‌تر قرار می‌گیرد). اگر معماری کامپیوتر دریافت کننده این پیام Big Endian باشد، باید ابتدا ترتیب بایت های این فیلدها را برعکس کند و سپس آن‌ها را به عنوان یک عدد تفسیر کند.

با توجه به ساختار پیام، میان‌افزار Quax می‌تواند در هر بسته ارسالی حداکثر 256 عدد Job و 4 گیگابایت اطلاعات (Payload) قرار دهد.

4. تشریح عملکرد Workerها

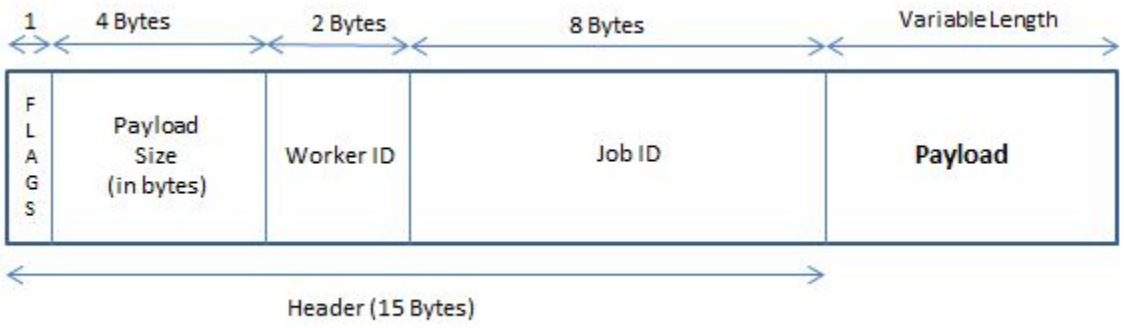
هر Worker بعد از اینکه خود را به میان‌افزار معرفی کرد به صورت پیش‌فرض به پورت شماره 20000 گوش می‌دهد و منتظر رسیدن یک مجموعه از Jobها می‌شود. وقتی بسته جدیدی دریافت شد، Worker ابتدا Job های درون آن را به انتهای صفی به نام صف دریافت (Receive Queue) اضافه می‌کند. یک نخ به نام Worker Thread وظیفه دارد که Jobها را به ترتیب از ابتدای صف خارج کند، آنها را اجرا کند و نتایج را به میان‌افزار ارسال کند.

این بخش از Worker را که وظیفه آن دریافت Job، اجرای آن و ارسال نتایج است را Worker Agent نام‌گذاری کرده‌ایم.

هر Worker دارای یک واحد کنترل نیز می‌باشد. واحد کنترل وظیفه دارد پیام‌های کنترلی را از میان‌افزار دریافت کند و آن‌ها را اجرا کند (پیام‌هایی مانند توقف یا شروع مجدد و غیره). همچنین واحد کنترل وظیفه ثبت Worker را برعهده دارد (اضافه کردن Worker به صف Available Workers).

4.1. ساختار اطلاعات ارسال شده توسط Worker

بعد از اینکه Worker Agent یک Job را اجرا کرد، نتایج حاصل از اجرای آن Job در یک فایل ذخیره می‌شود. Worker Agent باید محتویات این فایل را به واحد دریافت کننده نتایج ارسال کند. ساختار پیامی که Worker Agent ارسال می‌کند به صورت زیر است:



فیلد Flags رزرو شده است.

فیلد Payload Size: اندازه payload (بر حسب بایت). اندازه Payload حداکثر می‌تواند 4 گیگابایت باشد.

فیلد Worker ID: شناسه Workerای که این بسته را ارسال کرده است. هر Worker باید شناسه خودش را در این فیلد قرار دهد.

فیلد Job ID: شناسه Job ای که اجرا شده و نتایج آن در قسمت Payload قرار دارد.

Payload: حاوی نتایج تولید شده توسط Job است (اطلاعات فایل خروجی تولید شده توسط Job)

5. واحد دریافت کننده نتایج

وظیفه این واحد دریافت نتایج از Worker ها و تحویل آن به برنامه کاربردی است. همچنین این واحد با رسیدن نتایج یک job، واحد گسیل کننده Job را نیز از این موضوع مطلع می‌کند تا بتواند جدول Pending Jobs خود را به روز رسانی کند. واحد دریافت کننده نتایج به صورت پیش‌فرض روی پورت با شماره 40000 منتظر دریافت نتایج Job ها است.

6. واحد کنترل

هم Job Server و هم Workerها دارای یک واحد به نام واحد کنترل می‌باشند. Worker و Job Server از طریق این واحد می-توانند به هم پیام‌های کنترلی ارسال کنند. پیام های از قبیل "اعلام زنده بودن" ، "درخواست توقف Worker" ، "درخواست شروع مجدد Worker" ، "درخواست ثبت Worker" و غیره.

```
//
// Class Job. This class defines a job.
// only public members of this class are shown.
//

public class Job
{
     // Properties
     public ulong ID { get; set; }
     public string ExecuteableFileName { set; get; }
     public byte[] InputFileContent { set; get; }
     public TimeSpan ExpectedTimeToComplete { set; get; }
     public TimeSpan MaxTimeToComplete { set; get; }
     public long GetTotalLength()
}
```

```
//
// Class Quax. This is the main Quax class that creates and manages the middleware.
// only public members of this class are shown.
//

public class Quax
{
     // Types
     public delegate void JobResultReciveEventHandler(JobResult result);
     public delegate void NewLogItemEventHandler(Log log);


     // Events
     public event JobResultReciveEventHandler JobResultRecive;
     public event NewLogItemEventHandler NewLogItem;


     // Properties
     public int MaxDispatcherThreads;
     public int DataChannelPortNumber; // Job server listens to this port for incoming job
     results.

     public int ControlChannelPortNumber; // Job server listens to this port for incoming
     control messages.

     public bool IsDispatching;
     public Version AssemblyVersion;
```

```
    // Methods

    // Adds an job to the end of the jobs queue.
    public void AddJob(Job j)



    // Adds the elements of an array of jobs to the end of the send queue.
    public void AddJobs(Job[] jobs)


    // Removes all jobs from send queue.
    public void ClearJobsQueue()


    // Starts the process of dispatching jobs to the workers for computing.
    public void StartDispatch()


    // Stops the process of dispatching jobs. the process can be resumed by calling
    StartDispatch().

    public void StopDispatch()

    // Shutdown quax object
    public void Shutdown()

}
```

```
//
// Class Log. (this log are generated by Quax object and are reported to application.
//
public class Log
{
      public string Message { set; get; }

      public class JobDispatcherStartedLog : Log
      {}
      public class JobDispatcherCouldNotStartLog : Log
      {}
      public class JobDispatcherStartedLog : Log
      {}
      public class JobDispatcherStoppedLog : Log
      {}
      public class JobDispatcherShutdownLog : Log
      {}
      public class BatchOfJobsSentLog : Log
      {}
      public class JobTooBigLog: Log
      {}
      public class JobExecutableFileReadErrorLog: Log
      {}
```

```
	public class UnresponsiveWorkerLog: Log
	{}

	public class WorkerRegisterationRequestLog: Log
	{}
	public class ErrorInSendingJobToWorkerLog: Log
	{}
}
```